第三章 相关关系

记录人：241601051

3.1 认识相关关系、函数关系和因果关系

3.1.1 函数关系

3.1.1.1 函数关系定义

函数关系是指变量之间存在的严格确定性依存关系，即当一个或几个变量取一定值的时，另一个变量有唯一确定的值与之对应。即在一定条件下，给定自变量x的值时，因变量y 由一个明确的数学规则或公式决定。例如，圆的面积A 与半径 r 之间存在函数关系 $A = \pi r^2$，给定任意一个半径值，都能唯一确定对应的面积值；物理学中的自由落体运动公式$h = \frac{1}{2}gt^2$，其中 h 表示物体下落的高度，t 表示下落的时间，g 是重力加速度。在这个公式中，给定任意一个时间 t 的值，都能通过这个函数关系唯一地计算出物体下落的高度 h，这就是一个典型的函数关系。在函数关系中，自变量和因变量之间有着明确的数学表达式 $y = f（x）$，因变量 y 的变化完全依赖于自变量 x 的变化，不存在任何随机性。

3.1.1.2 函数关系特征

（1）确定性：每个自变量 x 对应唯一的因变量y，即给定x 值，y 的值是确定的。函数关系不存在多值性，也就是同一个自变量x不会映射到多个不同的因变量y上。

（2）单值性：对于每一个输入 x，存在唯一的输出 y 。这意味着 y 只能是由 x 通过函数关系得出的一个值，无法有多个可能的结果。例如，在线性函数 y=2x+1 中，给定 x=3时，y 的值为 y=2*3+1=7，是唯一的。

（3）可表达性：函数关系可以通过数学表达式进行明确的描述。这使得我们可以借助数学工具进行分析、推理、计算，甚至进行推断和预测。

3.1.1.3函数关系的应用

函数关系在数据分析中应用广泛，特别是在回归分析和预测模型的建立中。通过理解并利用自变量和因变量之间的函数关系，分析师能够构建预测模型、揭示数据趋势，并做出科学决策。

（1）回归分析的 Python 实现

回归分析是通过一个数学函数来拟合自变量和因变量之间的关系，常见的包括线性回归、二次回归、指数回归等。

1）线性回归

这是最常见的回归分析，适用于因变量与自变量之间具有线性关系的情况。可以使用 `scikit-learn` 库实现。

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

# 生成一些示例数据
x = np.array([1, 2, 3, 4, 5]).reshape(-1, 1)
y = np.array([2, 4, 5, 4, 5])

# 创建并训练线性回归模型
```

```
model = LinearRegression()
model.fit(x, y)

# 预测值
y_pred = model.predict(x)

# 可视化结果
plt.scatter(x, y, color='blue', label='实际数据')
plt.plot(x, y_pred, color='red', label='拟合线')
plt.xlabel('自变量 x')
plt.ylabel('因变量 y')
plt.legend()
plt.title('线性回归示例')
plt.show()

# 输出拟合的系数
print(f"斜率 (a): {model.coef_[0]}")
print(f"截距 (b): {model.intercept_}")
```

2）二次回归

当数据呈现非线性关系时，可以使用二次回归模型。

```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression

# 生成二次曲线的数据
x = np.array([1, 2, 3, 4, 5]).reshape(-1, 1)
y = np.array([1, 4, 9, 16, 25])

# 进行二次回归
poly = PolynomialFeatures(degree=2)
x_poly = poly.fit_transform(x)
model = LinearRegression()
model.fit(x_poly, y)

# 预测值
y_pred = model.predict(x_poly)

# 可视化结果
plt.scatter(x, y, color='blue', label='实际数据')
plt.plot(x, y_pred, color='red', label='拟合曲线')
plt.xlabel('自变量 x')
plt.ylabel('因变量 y')
plt.legend()
plt.title('二次回归示例')
plt.show()

# 输出拟合的系数
print(f"拟合的系数: {model.coef_}")
```

3）岭回归

岭回归是一种用于处理多重共线性问题的回归方法，常用于高维数据分析

```python
from sklearn.linear_model import Ridge

# 生成示例数据
x = np.array([[1], [2], [3], [4], [5], [6]])
y = np.array([3, 6, 7, 8, 10, 12])

# 使用岭回归
ridge = Ridge(alpha=1)
ridge.fit(x, y)

# 预测值
y_pred = ridge.predict(x)

# 可视化结果
plt.scatter(x, y, color='blue', label='实际数据')
plt.plot(x, y_pred, color='red', label='岭回归拟合线')
plt.xlabel('自变量 x')
plt.ylabel('因变量 y')
plt.legend()
plt.title('岭回归示例')
plt.show()

# 输出岭回归系数
print(f"岭回归系数: {ridge.coef_}")
```

（2）数据拟合的python实现

除了回归分析，函数关系还常用于数据拟合，尤其是在实验数据中。以下是几个常见的数据拟合方法

1）指数回归拟合

假设我们有一组数据，可以通过指数函数 $y = Ae^{-kx}$ 来拟合。以下是用 Python 进行指数回归拟合的示例：

```python
from scipy.optimize import curve_fit

# 生成示例数据 (模拟衰减过程)
def exp_func(x, A, k):
    return A * np.exp(-k * x)

x_data = np.linspace(0, 4, 50)
y_data = 3 * np.exp(-1.2 * x_data) + np.random.normal(0, 0.2, x_data.size)
 # 添加噪声

# 使用 curve_fit 进行拟合
params, covariance = curve_fit(exp_func, x_data, y_data, p0=[1, 1])

# 可视化结果
plt.scatter(x_data, y_data, color='blue', label='实际数据')
plt.plot(x_data, exp_func(x_data, *params), color='red', label='拟合曲线')
plt.xlabel('时间 x')
plt.ylabel('反应速率 y')
```

```
plt.legend()
plt.title('指数衰减拟合示例')
plt.show()

# 输出拟合参数 A 和 k
print(f"拟合参数 A: {params[0]}")
print(f"拟合参数 k: {params[1]}")
```

2）高斯回归拟合

对于一些具有峰值的数据，可以使用高斯函数进行拟合。高斯回归通常用于信号处理和统计分析。

```
# 高斯函数定义
def gaussian(x, A, mu, sigma):
    return A * np.exp(-(x - mu)**2 / (2 * sigma**2))

x_data = np.linspace(-5, 5, 100)
y_data = gaussian(x_data, 1, 0, 1) + np.random.normal(0, 0.1, x_data.size)  # 添加噪声

# 使用 curve_fit 进行高斯拟合
params, covariance = curve_fit(gaussian, x_data, y_data, p0=[1, 0, 1])

# 可视化结果
plt.scatter(x_data, y_data, color='blue', label='实际数据')
plt.plot(x_data, gaussian(x_data, *params), color='red', label='拟合曲线')
plt.xlabel('自变量 x')
plt.ylabel('因变量 y')
plt.legend()
plt.title('高斯回归拟合示例')
plt.show()

# 输出拟合参数 A, mu 和 sigma
print(f"拟合参数 A: {params[0]}")
print(f"拟合参数 mu: {params[1]}")
print(f"拟合参数 sigma: {params[2]}")

```

3.1.2 因果关系

3.1.2.1 因果关系定义

因果关系指的是一种直接的、确定的影响过程，其中一个事件或现象（因）引起另一个事件或现象（果）。换句话说，因果关系强调的是“因”对“果”的作用，而这种作用在时间上通常是单向的。因果关系通常通过实验设计或者统计方法来验证和建立。例如，假设我们观察到变量 x（自变量）和 y（因变量）之间存在关系。如果我们能够证明， x的改变直接引起了 y 的改变，并且这个改变不是由于其他第三方变量的影响，那么我们可以说， x 和 y 存在因果关系。

因果关系的经典表述：

- **A是B的原因**，意味着当A发生时，B也会发生；
- **A引起B**，意味着A的变化会导致B的变化；
- **A导致B**，意味着A发生后，B随之变化。

3.1.2.2 因果关系的特征

（1）时间顺序性：因果关系要求“因”先于“果”发生。即在时间上，因变量的变化不能先于自变量的变化。

（2）相关性：尽管因果关系并非仅仅依赖于相关性，但通常，如果存在因果关系，那么因变量和自变量之间会有一定的相关性。也就是说，因果关系是相关性的前提条件，但相关性并不必然意味着因果关系。

（3）排除其他因素：因果关系的验证通常要求排除其他潜在的影响因素。这一特征要求我们要确保其他可能导致结果的变量不干扰因果关系的推断。通过控制变量或者进行实验设计，可以尽量消除这些干扰。

（4）可重复性和普适性：理想的因果关系应当具有可重复性和普适性，即它在不同的样本、时间和环境中应该是稳定的。

3.1.2.3 因果关系的应用

因果关系广泛应用于多个领域，如医学研究、社会科学、经济学、物理学等。在数据分析中，因果关系的发现不仅帮助我们理解现象，还能为预测和决策提供依据。常见的应用场景包括：

- **医疗领域**：通过临床实验或观察性研究，我们可以找出某种药物是否有效，或者某种生活方式是否会导致健康问题。例如，研究发现吸烟与肺癌之间存在因果关系，吸烟是导致肺癌的一个重要原因。
- **经济学**：经济政策的制定往往依赖于对因果关系的理解。例如，研究表明，降低利率可能会导致消费者增加支出，从而刺激经济增长。
- **社会科学**：因果推断可以帮助理解社会现象背后的原因，比如教育水平与收入水平之间的因果关系。

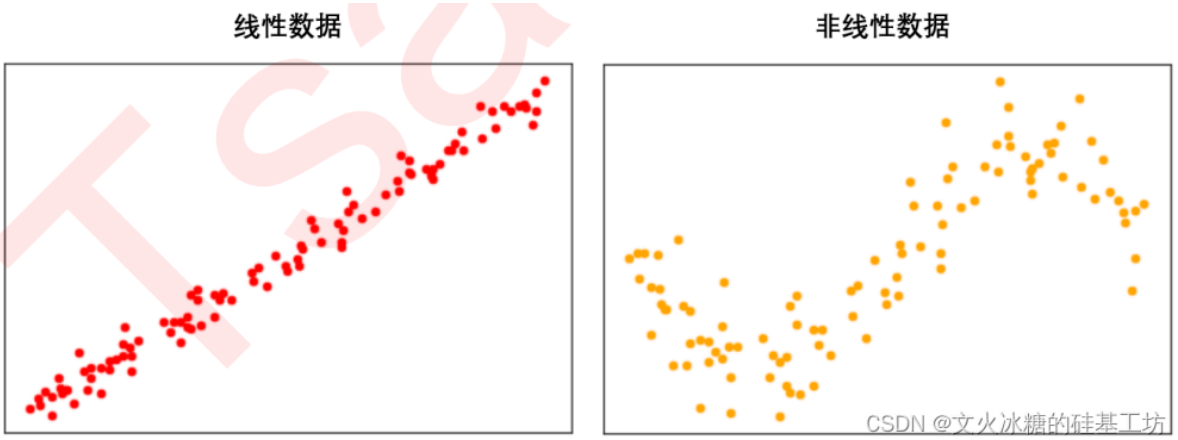
3.1.3 相关关系

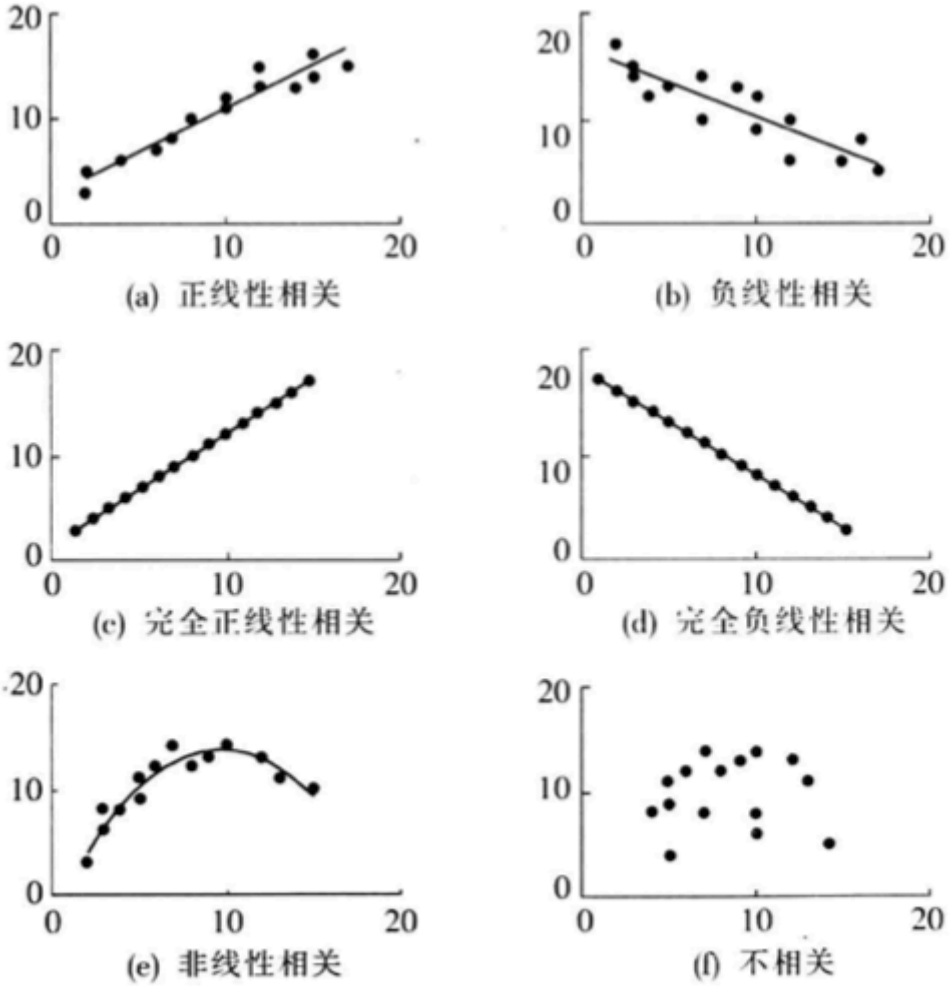
3.1.3.1 相关关系定义

相关关系是指变量之间存在一种非确定依存关系，即变量间的关系不能用函数表达式精准表达。由于受随机因素影响，一个变量的值不能用另一个变量唯一确定；衡量两个变量之间的相关关系强弱程度的统计指标即是相关系数；根据变量相关的方向，可以将相关关系分为正相关和负相关。正相关即是指一个变量增加的同时另一个变量也随之增加，负相关即是指当一个变量增加时，另一个变量相应减少。

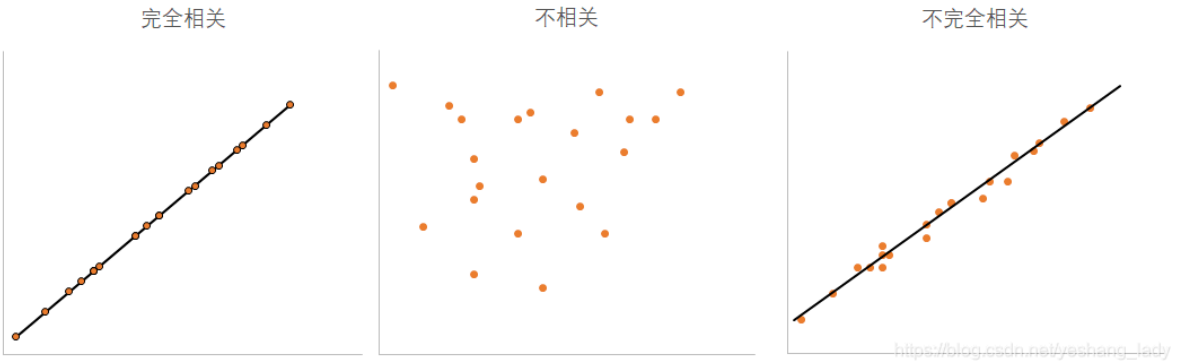
3.1.3.2 相关关系分类

（1）线性相关和非线性相关：线性相关是指两个或多个变量之间存在线性关系，可以通过一条直线来描述他们之间的关系。非线性相关是指自变量和因变量不呈线性关系，而是呈现曲线、抛物线关系或为不定量。





（2）完全相关、不完全相关和不相关：完全相关是指两个变量之间存在线性关系，且其相关系数为+1或-1，在完全相关的情形下，两个变量的散点图呈现一条直线。斜率为正为正相关，斜率为负为负相关。



（3）单相关和复相关：根据变量个数可分为单相关和复相关。单相关是指两个变量之间的相关关系，也称为一元相关。负相关是指三个及以上变量之间的相关关系，也成为多元相关。

3.1.3.3 相关关系特征

（1）对称性：相关系数是对称的，即 $Cov(X,Y) = Cov(Y,X)$。这说明无论哪个变量作为自变量或因变量，相关性是相同的。

（2）敏感性：相关系数对异常值（离群点）非常敏感。极端的离群点可能会显著改变相关系数的值，从而影响分析的准确性。

3.1.3.3 相关系数

相关系数是衡量两个变量之间相关关系强弱程度的统计指标。相关系数（Correlation Coefficient）通常用符号 r 表示，用于衡量两个变量 X 和 Y 之间的线性相关程度。它的取值范围在 -1 到 1 之间，即 $-1 \le r \le 1$。

- **r=1**：表示完全正相关，即一个变量增加时，另一个变量也按比例增加。
- **r=−1**：表示完全负相关，即一个变量增加时，另一个变量按比例减少。
- **r=0**：表示两个变量之间没有线性相关关系。
- **|r| 接近 1**：表示相关性很强。
- **|r| 接近 0**：表示相关性很弱。

相关系数的特点：

（1）对称性：相关系数r不区分自变量和因变量，$r_{xy}=r_{yx}$；

（2）无量纲性：相关系数是一个无量纲的量，不受变量单位的影响；

（3）线性相关性：相关系数衡量的是线性相关程度，不能反映非线性关系。如果两个变量之间存在非线性关系，相关系数可能接近于零；

（4）样本相关系数的局限性：样本相关系数 r 是对总体相关系数 ρ 的估计，可能会受到样本大小和抽样误差的影响。

3.1.4 三者之间的区别与联系

3.1.4.1 三者之间的区别：

- **确定性程度**：函数关系具有完全的确定性，给定自变量的值，因变量的值就能唯一确定；因果关系虽然也有一定的确定性，但这种确定性是基于逻辑和实验验证的，可能存在多种因素共同作用导致结果的发生；相关关系则是一种统计意义上的关联，具有不确定性，即使两个变量之间存在相关关系，也不能确定一个变量的变化一定会引起另一个变量的变化。
- **表现形式**：函数关系通常可以用明确的数学公式或函数模型来表示，如 y=f(x)；因果关系则需要通过因果图、因果模型等来描述事件之间的逻辑联系；相关关系主要通过相关系数、散点图等统计方法来展示变量之间的关联程度和趋势。
- **研究方法**：函数关系的研究侧重于数学建模和解析方法，通过建立数学模型来描述变量之间的关系，并进行精确的计算和预测；因果关系的研究需要运用实验设计、观察研究、因果推断等方法，通过控制变量、排除干扰因素等手段来确定因果联系；相关关系的研究则主要依赖于统计分析方法，如相关系数计算、回归分析等，通过对数据的统计分析来揭示变量之间的关联。
- **应用场景**：函数关系广泛应用于自然科学、工程技术等领域，用于描述各种确定性的物理、化学、生物等过程；因果关系在医学、社会科学、经济学等领域的研究中尤为重要，用于探究现象背后的因果机制，为政策制定、干预措施等提供依据；相关关系则在数据分析、市场研究、风险评估等众多领域都有应用，用于发现变量之间的潜在关联，为决策提供参考。

3.1.4.2 三者之间的联系：

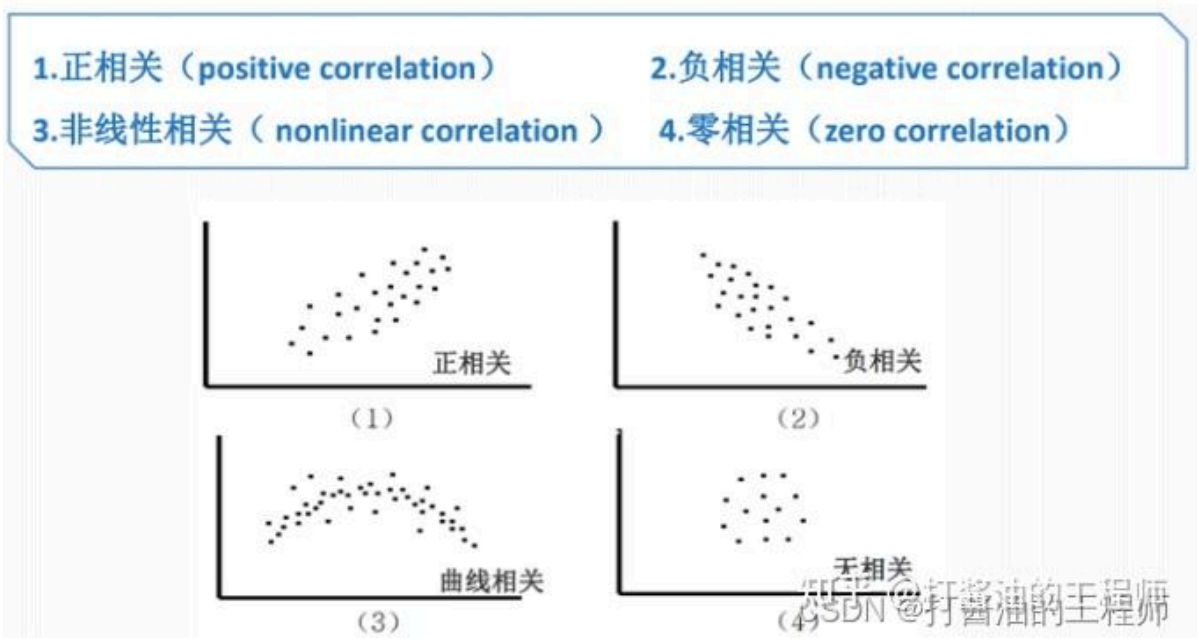
- **函数关系与相关关系**：函数关系可以看作是相关关系的一种特殊形式。当两个变量之间存在函数关系时，它们之间必然存在相关关系，且相关程度通常较强。例如，圆的周长 C 与直径 d 之间存在函数关系 C=πd，从相关分析的角度来看，周长和直径之间也存在完全的正相关关系。然而，并非所有相关关系都是函数关系，相关关系还包括那些非确定性的、非线性的关联。

- **因果关系与相关关系**：因果关系往往表现为相关关系，但相关关系并不一定意味着因果关系。当两个变量之间存在因果关系时，它们之间通常会表现出一定的相关性。例如，吸烟与肺癌之间存在因果关系，从统计数据上也可以看到吸烟率与肺癌发病率之间存在相关关系。但是，仅仅发现两个变量之间存在相关关系，并不能直接得出它们之间存在因果关系的结论，因为可能存在其他干扰因素或潜在的第三变量同时影响这两个变量，导致它们之间出现相关性。例如，冰淇淋销量与溺水事故之间可能存在相关关系，但这并不是因为吃冰淇淋导致溺水，而是因为它们都与夏季高温这一第三变量有关。因此，相关关系可以为寻找因果关系提供线索和初步的证据，但需要进一步的分析和验证才能确定因果关系。
- **函数关系与因果关系**：在某些情况下，函数关系可以用来描述因果关系。当因果关系可以用明确的数学模型来表示时，这种模型就可以看作是一种函数关系。例如，在物理学中，力与加速度之间的关系可以用牛顿第二定律 F=ma 来描述，这个公式既是一个函数关系，也体现了力对加速度的因果影响。然而，并非所有的因果关系都能用简单的函数关系来描述，特别是在社会科学和复杂系统中，因果关系往往更加复杂，涉及到多个因素的相互作用和非线性关系。

3.2 线性相关分析

3.2.1 认识线性相关分析

线性相关分析是一种用于探索两个变量之间线性关系强度与方向的统计方法。在统计学和数据科学中，线性关系广泛应用于建模、预测以及变量间关系的理解。相关分析通过计算相关系数来度量变量间的线性依赖程度，从而判断它们是否存在线性关系。如果变量之间存在线性关系，它们的散点图大致如下图。



线性相关分析的步骤：

1.明确研究问题

确定需要分析的两个变量 X和 Y，并明确它们之间的关系是否具有研究意义。例如：

- 经济学中，研究收入与消费的关系。
- 生物学中，研究基因表达与疾病发生的关系。

2.数据收集

收集足够的样本数据，确保数据的准确性和代表性。数据可以是实验数据、调查数据或观测数据。

3. 数据可视化

绘制散点图，观察变量之间的关系是否呈线性分布。散点图是线性相关分析的重要工具，可以帮助直观判断变量之间的关系。

- **正线性相关**：数据点从左下角到右上角分布。
- **负线性相关**：数据点从左上角到右下角分布。
- **无明显线性关系**：数据点随机分布，无明显趋势。

4.计算相关系数

相关系数是衡量线性相关程度的量化指标，最常用的是**Pearson相关系数（详见3.2.3）、Speaman相关系数（详见3.2.4）、Kendall相关系数（详见3.2.5）**

5. 解释相关系数

根据计算得到的相关系数 r ，判断变量之间的线性相关程度和方向，例如Pearson相关系数 r ：

- **强相关**：$\mid r \mid > 0.7$
- **中等相关**：$0.3 < \mid r \mid \leq 0.7$
- **弱相关**：$\mid r \mid \leq 0.3$

6. 假设检验

为了判断相关系数是否在统计学上显著，通常需要进行假设检验。常用的假设检验方法是**t检验**，其公式为：

$$t = \frac{r\sqrt{n-2}}{1-r^2}$$

其中：

- r 是样本相关系数。
- n 是样本数量。

根据 t 值和自由度$df = n - 2$，查找t 分布表，确定p 值。如果p 值小于显著性水平（如$\alpha = 0.05$或$\alpha = 0.1$），则认为两个变量相关。

7. 结果解释与应用

根据相关系数和假设检验的结果，解释变量之间的关系，并结合实际背景进行讨论。例如：

- 如果相关系数显著且为正，说明两个变量之间存在正线性关系。
- 如果相关系数不显著，说明两个变量之间可能不存在线性关系。

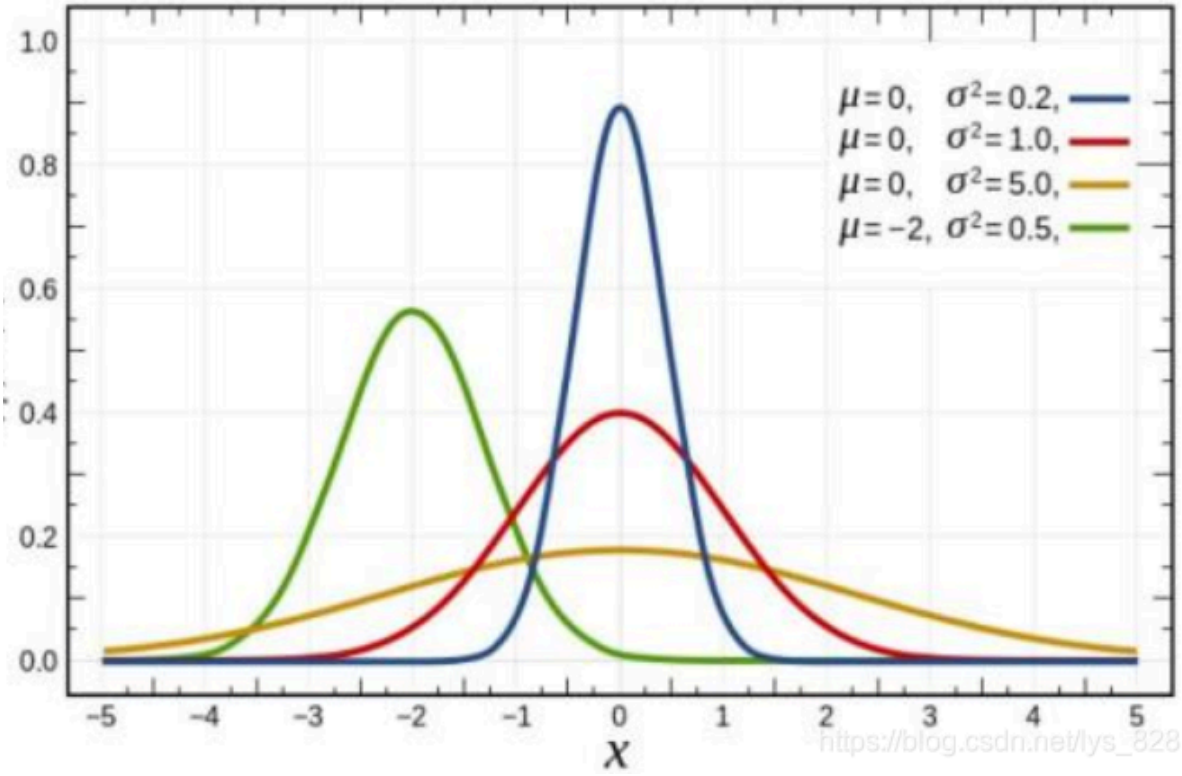
3.2.2 相关分析的正态性检验

3.2.2.1正态分布

正态分布（Normal Distribution），也称为高斯分布（Gaussian Distribution），是一种连续型概率分布。其概率密度函数为：

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

其中，μ是均值（期望值），σ 是标准差，σ^2 是方差。正态分布的图形是一个关于 μ 对称的钟形曲线。



正态分布特征：

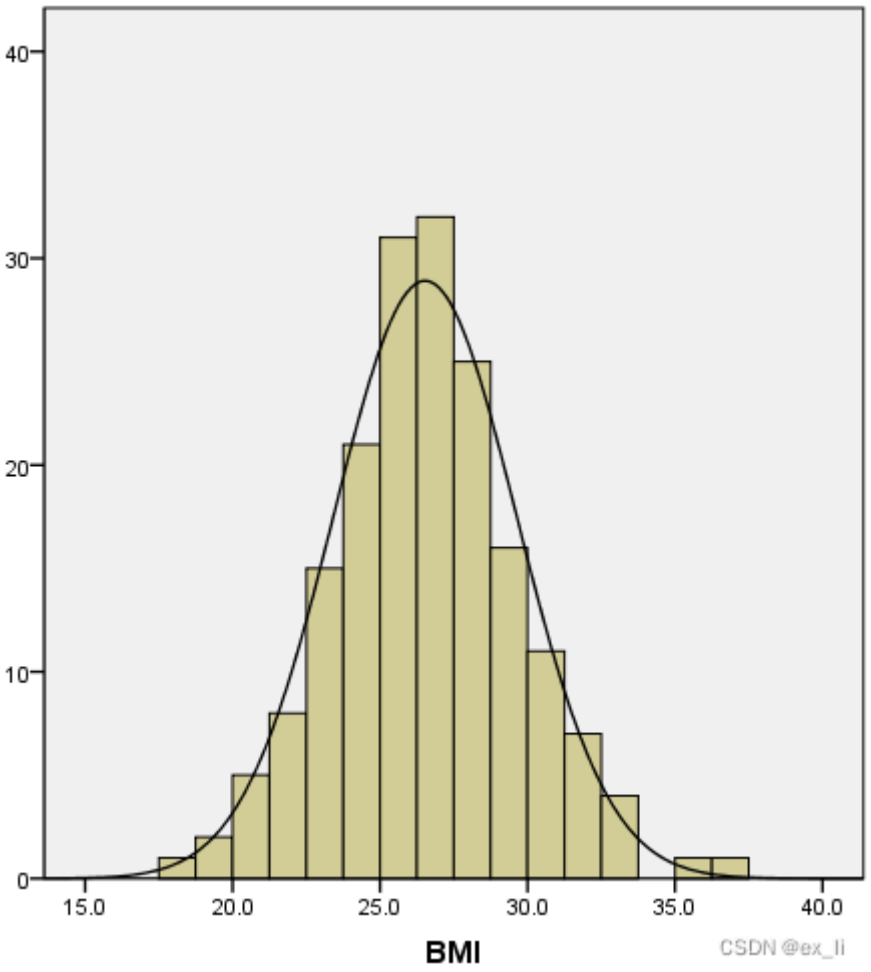
- **对称性**：正态分布曲线关于均值 μ 对称，即$f(\mu + x) = f(\mu - x)$。
- **集中趋势**：数据集中在均值附近，越远离均值，概率密度越小。
- **“3σ”原则**：
  - 约 68.27% 的数据落在$(\mu - \sigma, \mu + \sigma)$ 区间内；
  - 约 95.45% 的数据落在 $(\mu - 2\sigma, \mu + 2\sigma)$ 区间内；
  - 约 99.73% 的数据落在$(\mu - 3\sigma, \mu + 3\sigma)$区间内;
- **均值、中位数和众数相等**：在正态分布中，均值、中位数和众数都等于μ。

3.2.2.2正态性检验

在进行线性分析之前，应先了解数据是否符合正态分布。常用的正态性检验方法有两种方法：图示法和假设检验法.正态检验的图示法包括直方图、P-P图和Q-Q图法

3.2.2.2.1直方图

直方图通过将数据划分为不同的区间来展示数据分布情况，每个箱子的高度表示该区间点的数量或频率。符合正态分布的直方图的顶端曲线相连后是一条呈钟形的曲线，直方图的左右越均衡，其正态性越强。



直方图的绘制步骤如下：

1. 准备数据

- 收集或生成样本数据。
- 确保数据是数值型且已清洗（去除缺失值、异常值等）。

2. 确定桶数和桶宽

- **桶数（bin count）**：选择合适的桶数，常见的方法包括：
  - **Sturges公式**：$\text{bins} = \lceil \log_2(n) + 1 \rceil$,其中 n 是数据点的数量。
  - **平方根法则**：$\text{bins} = \sqrt{n}$
  - ** Freedman-Diaconis规则**：$\text{bin width} = 2 \times \text{IQR} \times n^{-1/3}$，其中$IQR$是四分位距。
- **桶宽（bin width）**：根据数据范围和桶数计算桶宽：

$$\text{bin width} = \frac{\max(x) - \min(x)}{\text{bins}}$$

3. 计算频率

- 对数据进行分组，统计每个桶内的数据点数量。
- 可以选择显示绝对频率或相对频率。

4. 绘制直方图

```
import numpy as np
import matplotlib.pyplot as plt
```

```python

# 生成样本数据（正态分布）
np.random.seed(0)
data = np.random.normal(loc=0, scale=1, size=1000)  # 均值为0，标准差为1的正态分布数据

# 确定桶数和桶宽
# 使用Sturges公式计算桶数
n = len(data)
bins = int(np.ceil(np.log2(n)) + 1)
bin_width = (max(data) - min(data)) / bins

# 绘制直方图
plt.figure(figsize=(10, 6))
plt.hist(data, bins=bins, edgecolor='black', alpha=0.7, color='blue', density=True)  # density=True表示显示相对频率
plt.title('Histogram of Normal Distribution Data')
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.grid(axis='y', linestyle='--', alpha=0.7)

# 添加核密度估计（KDE）曲线
from scipy.stats import gaussian_kde
kde = gaussian_kde(data)
x_range = np.linspace(min(data), max(data), 1000)
plt.plot(x_range, kde(x_range), color='red', linewidth=2, label='KDE')

plt.legend()
plt.show()
```

5.图形解释

(1) 形状:

- 正态分布的直方图呈钟形，即中间高、两边低，对称于均值。

(2) 对称性:

- 正态分布的直方图关于均值对称，即左右两边的形状和面积相等。如果直方图明显不对称，说明数据可能不服从正态分布。

(3) 偏态:

- 如果直方图的尾部向右延伸，称为右偏（正偏）。
- 如果直方图的尾部向左延伸，称为左偏（负偏）。
- 偏态的存在说明数据可能不服从正态分布。

(4) 峰度:

- 正态分布的直方图具有适中的峰度，即峰值既不过高也不过低。
- 如果直方图的峰值过高，称为尖峰（leptokurtic）。
- 如果直方图的峰值过低，称为平峰（platykurtic）。
- 峰度的异常说明数据可能不服从正态分布。

3.2.2.2.2 Q-Q图

Q-Q图的原理是检验实际分位数与理论分位数是否吻合。若吻合，则散点应围绕在一条直线周围，或者实际分位数与理论分位数之差分布在对称于以0为水平轴的带内。

Q-Q图的绘制步骤

以下是绘制Q-Q图的具体步骤，以检验数据是否服从正态分布为例：

1. 准备数据

- 收集或生成样本数据。
- 确保数据是数值型且已清洗（去除缺失值、异常值等）。

2. 计算样本数据的分位数

- 对样本数据进行排序。
- 计算每个数据点的分位数（百分位数）。通常使用公式：

$$p_i = \frac{i - 0.5}{n}$$

  其中，i 是数据点的索引（从1到 n），n 是样本数据的总数。

3. 计算理论分布的分位数

- 对于正态分布，根据分位数 p_i，计算对应的理论分位数 Z_i：

$$z_i = \Phi^{-1}(p_i)$$

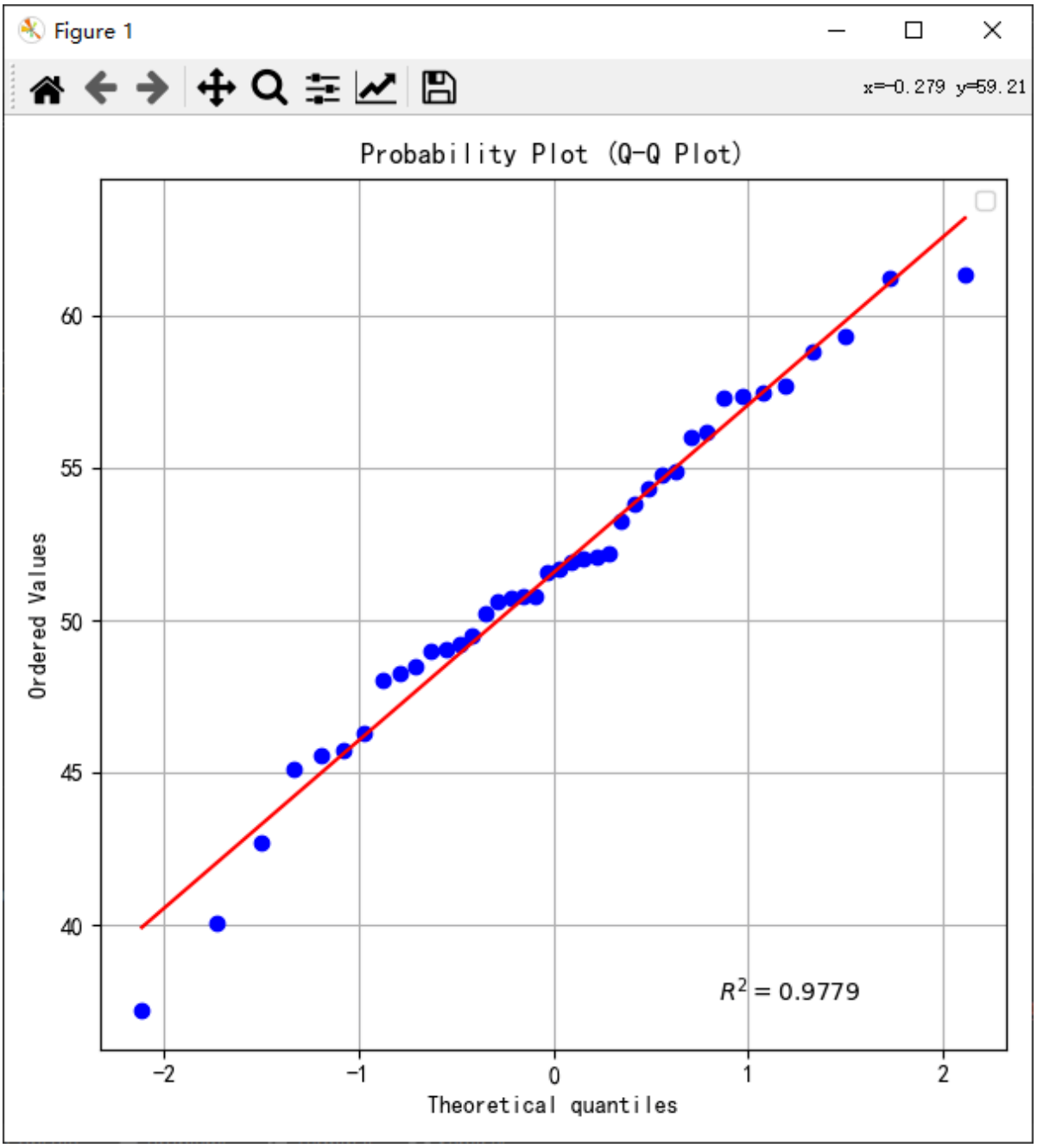
  其中，Φ^{-1}是标准正态分布的累积分布函数（CDF）的逆函数。

4. 绘制Q-Q图

- 在Q-Q图上，横轴表示理论分位数 Z_i，纵轴表示样本数据的分位数对应的值。
- 如果样本数据服从正态分布，Q-Q图上的点应接近于一条直线。

5.图形解释：

- **理想情况**：如果样本数据完全服从理论分布，Q-Q图上的点将紧密地落在一条直线上（如下图）。



- **偏离情况**：如果样本数据不服从理论分布，Q-Q图上的点会偏离直线，形成不同的形状。例如：
  - **弯曲的曲线**：可能表示数据存在偏态或峰度异常。
  - **S形曲线**：可能表示数据存在重尾（leptokurtic）或轻尾（platykurtic）问题。
  - **离群点**：可能表示数据中存在异常值。

Q-Q图的Python实现

```python
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.api as sm

# 生成样本数据（正态分布）
np.random.seed(0)
data = np.random.normal(loc=0, scale=1, size=1000)  # 均值为0，标准差为1的正态分布数据

# 绘制Q-Q图
```

```
fig = sm.qqplot(data, line='45', fit=True)  # line='45'表示绘制45度参考线，fit=True自动拟合数据
plt.title('Q-Q Plot for Normal Distribution')
plt.xlabel('Theoretical Quantiles')
plt.ylabel('Sample Quantiles')

# 显示图形
plt.show()
```

3.2.2.2.3 P-P图

P-P图以实际数据累积比例为X轴，标准正态分布（均值为0，标准差为1）的累积比例为Y，绘制散点图。P-P图的原理是检验实际累积概率分布与理论累积概率分布是否吻合。若吻合，则散点应围绕在一条直线周围，或者实际累积概率与理论累积概率之差分布在对称于以0为水平轴的带内。

P-P图的原理

1.累积分布函数（CDF）：

- 累积分布函数$F(x)$表示随机变量X小于或等于x的概率，即：

$$F(x) = P(X \leq x)$$

- 对于正态分布，理论累积分布函数可以通过标准正态分布的累积分布函数$\Phi(z)$来计算

2.P-P图的构造：

- P-P图通过比较样本数据的累积分布函数与理论分布的累积分布函数来判断数据是否符合该分布。
- 横轴表示理论分布的累积概率$\Phi(z)$，纵轴表示样本数据的累积概率$F(x)$。
- 如果样本数据完全服从理论分布，P-P图上的点将紧密地落在一条直线上。

3.图形解释：

- **理想情况**：如果样本数据完全服从理论分布，P-P图上的点将紧密地落在一条直线上。
- **偏离情况**：
  - **弯曲的曲线**：可能表示数据存在偏态或峰度异常。
  - **S形曲线**：可能表示数据存在重尾（$leptokurtic$）或轻尾（$platykurtic$）问题。
  - **离群点**：可能表示数据中存在异常值。

P-P图的绘制步骤

以下是绘制P-P图的具体步骤，以检验数据是否服从正态分布为例：

1. 准备数据

- 收集或生成样本数据。
- 确保数据是数值型且已清洗（去除缺失值、异常值等）。

2. 计算样本数据的累积概率

- 对样本数据进行排序。
- 计算每个数据点的累积概率 $F(x)$。通常使用公式：

$$F(x_i) = \frac{i}{n+1}$$

其中，i 是数据点的索引（从1到 n），n 是样本数据的总数。

3. 计算理论分布的累积概率

- 对于正态分布，根据样本数据的均值 μ 和标准差 σ，计算每个数据点的标准化值z_i：

$$z_i = \frac{x_i - \mu}{\sigma}$$

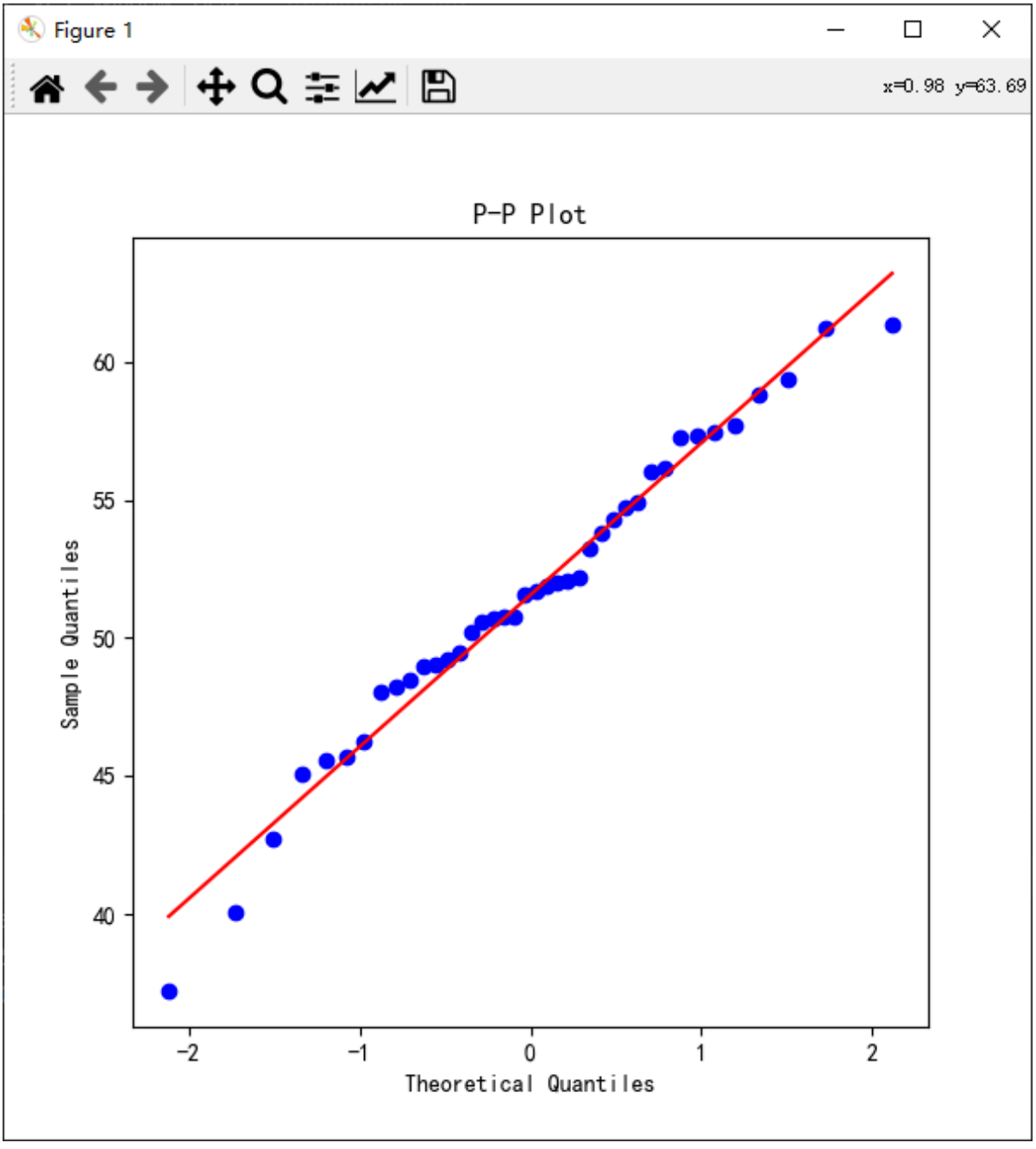
- 计算理论累积概率$\Phi(z_i)$，使用标准正态分布的累积分布函数。

4. 绘制P-P图

- 在P-P图上，横轴表示理论累积概率$\Phi(z_i)$，纵轴表示样本数据的累积概率$F(x_i)$。

5.图形解释

- 如果样本数据服从正态分布，P-P图上的点应接近于一条直线（如下图）。



P-P图的python实现

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm

# 生成样本数据（正态分布）
np.random.seed(0)
data = np.random.normal(loc=0, scale=1, size=1000)  # 均值为0，标准差为1的正态分布数据

# 计算样本数据的累积概率
n = len(data)
sorted_data = np.sort(data)
sample_cdf = np.arange(1, n + 1) / (n + 1)

# 计算理论分布的累积概率
z_values = (sorted_data - np.mean(data)) / np.std(data)
theoretical_cdf = norm.cdf(z_values)

# 绘制P-P图
plt.figure(figsize=(10, 6))
plt.plot(theoretical_cdf, sample_cdf, marker='o', linestyle='none', color='blue')
plt.plot([0, 1], [0, 1], color='red', linestyle='--')  # 参考线
plt.title('P-P Plot for Normal Distribution')
plt.xlabel('Theoretical CDF')
plt.ylabel('Sample CDF')
plt.grid(True)
plt.show()
```

3.2.2.2.4假设检验

正态性检验通常围绕以下假设进行：

- **原假设 (H_0)**：样本数据来自正态分布的总体。
- **备择假设 (H_1)**：样本数据不来自正态分布的总体。

根据检验结果，如果拒绝原假设 (H_0)，则认为数据不符合正态分布；如果不拒绝原假设，则认为数据可能符合正态分布。

常用的正态性检验方法

1.Shapiro-Wilk 检验

Shapiro-Wilk 检验是一种非常强大的正态性检验方法，特别适用于小样本数据（样本量$n < 5000$）。

检验步骤

1. **计算检验统计量 (W)**：
   - W 是样本数据的顺序统计量与正态分布的理论分位数之间的线性相关系数的平方。
   - W的值越接近 1，表示数据越接近正态分布。
2. **确定 p 值**：
   - 根据检验统计量 (W) 和样本量 (n)，通过查表或使用软件计算 p 值。
3. **做出判断**：

- 如果 p 值小于显著性水平（如 0.05），则拒绝原假设 (H_0)，认为数据不符合正态分布。
- 如果 p 值大于显著性水平，则不拒绝原假设，认为数据可能符合正态分布。

适用场景

- 小样本数据（$n < 5000$）。
- 对正态性检验非常敏感，适用于对正态性要求较高的统计分析。

2.Kolmogorov-Smirnov 检验

Kolmogorov-Smirnov 检验通过比较样本的累积分布函数（CDF）与理论正态分布的 CDF 来判断数据是否符合正态分布。

检验步骤

1. **计算检验统计量 (D)**:
   - (D) 是样本 CDF 与理论正态分布 CDF 之间的最大绝对差值。
2. **确定 p 值**:
   - 根据 (D) 和样本量 (n)，通过查表或使用软件计算 p 值。
3. **做出判断**:
   - 如果 p 值小于显著性水平（如 0.05），则拒绝原假设 (H_0)，认为数据不符合正态分布。
   - 如果 p 值大于显著性水平，则不拒绝原假设，认为数据可能符合正态分布。

适用场景

- 大样本数据（$n \geq 5000$）。
- 对数据的累积分布进行检验，适用于检查数据的整体分布特征。

3. Anderson-Darling 检验

Anderson-Darling 检验是一种基于数据与理论分布之间加权平方差的正态性检验方法。它对数据的尾部特征特别敏感。

检验步骤

1. **计算检验统计量 A^2**:
   - A^2 是样本数据与理论正态分布之间的加权平方差。
2. **确定 p 值**:
   - 根据A^2和样本量 n，通过查表或使用软件计算 p 值。
3. **做出判断**:
   - 如果 p 值小于显著性水平（如 0.05），则拒绝原假设 (H_0)，认为数据不符合正态分布。
   - 如果 p 值大于显著性水平，则不拒绝原假设，认为数据可能符合正态分布。

适用场景

- 适用于各种样本量。
- 对数据的尾部特征敏感，适用于需要严格检验正态分布的情况。

4. 如何选择正态性检验方法

1. **样本量大小**:
   - 小样本（$n < 5000$）：优先选择 Shapiro-Wilk 检验。

- 大样本（$n \geq 5000$）：优先选择 Kolmogorov-Smirnov 检验。

2. **数据特征**：
   - 如果数据的尾部特征重要，选择 Anderson-Darling 检验。
   - 如果需要快速判断整体分布，选择 Kolmogorov-Smirnov 检验。
3. **检验的敏感性**：
   - Shapiro-Wilk 检验对正态性非常敏感，适用于对正态性要求较高的分析。
   - Anderson-Darling 检验对尾部特征敏感，适用于严格检验正态分布的情况。

5.python实现示例

```python
import numpy as np
import scipy.stats as stats

# 示例数据
data = np.random.normal(loc=0, scale=1, size=100)  # 生成正态分布数据

# 1. Shapiro-Wilk 检验
shapiro_stat, shapiro_p = stats.shapiro(data)
print("Shapiro-Wilk 检验：")
print(f"统计量 W = {shapiro_stat:.4f}, p 值 = {shapiro_p:.4f}")
if shapiro_p < 0.05:
    print("拒绝原假设，数据不符合正态分布")
else:
    print("不拒绝原假设，数据可能符合正态分布")

# 2. Kolmogorov-Smirnov 检验
ks_stat, ks_p = stats.kstest(data, 'norm', args=(np.mean(data),
np.std(data)))
print("\nKolmogorov-Smirnov 检验：")
print(f"统计量 D = {ks_stat:.4f}, p 值 = {ks_p:.4f}")
if ks_p < 0.05:
    print("拒绝原假设，数据不符合正态分布")
else:
    print("不拒绝原假设，数据可能符合正态分布")

# 3. Anderson-Darling 检验
ad_stat, ad_critical, ad_significance = stats.anderson(data, dist='norm')
print("\nAnderson-Darling 检验：")
print(f"统计量 A^2 = {ad_stat:.4f}")
print("临界值：", ad_critical)
print("显著性水平：", ad_significance)
if ad_stat > ad_critical[2]:  # 通常使用 5% 显著性水平
    print("拒绝原假设，数据不符合正态分布")
else:
    print("不拒绝原假设，数据可能符合正态分布")
```

3.2.3 Person相关系数

3.2.3.1 Person相关系数的原理

3.2.3.1.1 定义：

- Person相关系数r 衡量两个变量 X 和 Y 之间的线性相关程度。
- 其取值范围为 [−1,1]：
  - $r = 1$：完全正相关，表示 X 增加时 Y 也增加。
  - $r = -1$：完全负相关，表示 X 增加时 Y 减少。
  - $r = 0$：无相关，表示 X和 Y 之间没有线性关系。
  - $|r| < 0.3$称为微弱相关
  - $0.3 \leq |r| \leq 0.5$称为微弱相关
  - $0.5 \leq |r| \leq 0.8$称为低度相关
  - $0.8 \leq |r| \leq 1$称为高度相关

3.2.3.1.2 公式：

$$r = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum_{i=1}^{n}(X_i - \bar{X})^2}\sqrt{\sum_{i=1}^{n}(Y_i - \bar{Y})^2}}$$

其中：

- X_i 和 Y_i 分别是变量 X 和 Y 的第 i 个观测值。
- $\bar{X}$和$\bar{Y}$分别是变量 X 和 Y 的样本均值。
- n是样本数量。

3.2.3.1.3 假设条件：

- 两个变量 *X* 和 *Y* 均应服从正态分布（或样本量较大时，根据中心极限定理可近似正态分布）。
- 两个变量之间的关系应为线性关系。
- 观测值应为成对数据，且相互独立。

3.2.3.2 Person相关系数计算步骤

1. 收集数据

- 收集两组变量 X 和 Y 的成对观测数据。

2. 计算均值

- 计算变量X和Y 的样本均值：

$$\bar{X} = \frac{1}{n}\sum_{i=1}^{n} X_i$$

$$\bar{Y} = \frac{1}{n}\sum_{i=1}^{n} Y_i$$

3. 计算协方差

- 计算 X 和 Y 的协方差：

$$\text{Cov}(X,Y) = \frac{1}{n}\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y}$$

4. 计算标准差

- 计算 X 和 Y 的标准差：

$$\sigma_X = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(X_i - \bar{X})^2}$$

$$\sigma_Y = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(Y_i - \bar{Y})^2}$$

5. 计算 Person相关系数

- 根据公式计算皮尔逊相关系数：

$$r = \frac{\text{Cov}(X,Y)}{\sigma_X \sigma_Y}$$

3.2.3.3 Person相关系数的python实现

```python
import numpy as np
from scipy.stats import pearsonr

# 生成示例数据
np.random.seed(0)
X = np.random.normal(loc=0, scale=1, size=100)  # 均值为0，标准差为1的正态分布数据
Y = 2 * X + np.random.normal(loc=0, scale=0.5, size=100)  # 假设Y与X线性相关，加上一些噪声

# 计算皮尔逊相关系数
r, p_value = pearsonr(X, Y)
print(f"Pearson Correlation Coefficient: {r:.4f}")
print(f"p-value: {p_value:.4f}")

# 手动计算皮尔逊相关系数
mean_X = np.mean(X)
mean_Y = np.mean(Y)
cov_XY = np.sum((X - mean_X) * (Y - mean_Y)) / len(X)
std_X = np.std(X)
std_Y = np.std(Y)
r_manual = cov_XY / (std_X * std_Y)
print(f"Manual Pearson Correlation Coefficient: {r_manual:.4f}")
```

3.2.4 Speaman秩相关系数

3.2.4.1 Speaman秩相关系数的原理

3.2.4.1.1 定义：

- 斯皮尔曼秩相关系数 ρ衡量两个变量 X 和 Y 之间的单调相关程度。
- 其取值范围为 [−1,1]：
  - $\rho = 1$：完全正相关，表示 X 增加Y也增加。
  - $\rho = -1$：完全负相关，表示 X 增加时Y减少。
  - $\rho = 0$：无相关，表示 X 和Y之间没有单调关系。

3.2.4.1.2 公式：

- 斯皮尔曼秩相关系数的计算公式为：

$$\rho = 1 - \frac{6 \sum_{i=1}^{n} d_i^2}{n(n^2 - 1)}$$

  其中：
  - d_i是变量 X和Y的第 i 个观测值的秩次差。
  - n 是样本数量。

3.2.4.1.3 假设条件：

- 两个变量 X 和 Y 应为有序变量（即可以进行秩次排列）。
- 观测值应为成对数据，且相互独立。

3.2.4.2 Speaman秩的计算步骤

以下是计算Speaman秩相关系数的具体步骤：

1. 收集数据

- 收集两组变量 X 和 Y 的成对观测数据。

2. 计算秩次

- 对变量 X 和 Y 的观测值分别进行秩次排列。
- 如果有相同的观测值，可以使用平均秩次。

3. 计算秩次差

- 计算每个观测值的秩次差d_i：

$$d_i = \text{rank}(X_i) - \text{rank}(Y_i)$$

4. 计算秩次差的平方和

- 计算秩次差的平方和：

$$\sum_{i=1}^{n} d_i^2$$

5. 计算Speaman秩相关系数

- 根据公式计算Speaman秩相关系数：

$$\rho = 1 - \frac{6\sum_{i=1}^{n} d_i^2}{n(n^2-1)}$$

3.2.4.3 Speaman秩的python实现

```python
import numpy as np
from scipy.stats import spearmanr

# 生成示例数据
np.random.seed(0)
X = np.random.rand(10)  # 生成10个随机数
Y = 2 * X + np.random.rand(10)  # 假设Y与X线性相关，加上一些噪声

# 计算Speaman秩相关系数
rho, p_value = spearmanr(X, Y)
print(f"Spearman's Rank Correlation Coefficient: {rho:.4f}")
print(f"p-value: {p_value:.4f}")
```

3.2.5 Kendall等级秩相关系数

3.2.5.1 原理

Kendall等级秩相关系数（Kendall's Tau）是一种用于衡量两个变量之间等级相关性的非参数方法。它通过比较数据对之间的排序关系来计算，适用于有序数据。Kendall's Tau 的取值范围为 [−1,1]，其中：

- $\tau = 1$ 表示完全正相关。
- $\tau = -1$ 表示完全负相关。
- $\tau = 0$ 表示无相关。

3.2.5.2 计算步骤

1.数据准备：

- 收集两组变量 X 和 Y的成对观测数据。

2.成对比较：

- 对于每对观测值$(X_i, Y_i)(X_i, Y_i)$和(X_j, Y_j)（其中$i < j$），比较它们的相对大小：
  - 如果 $X_i < X_j$且$Y_i < Y_j$，则该对被称为 **一致对。**
  - 如果$X_i > X_j$且$Y_i > Y_j$，则该对也被称为 **一致对。**
  - 如果$X_i < X_j$ 且$Y_i > Y_j$，则该对被称为 **不一致对。**
  - 如果$X_i > X_j$ 且$Y_i < Y_j$，则该对也被称为 **不一致对。**
  - 如果 $X_i = X_j$ 且$Y_i = Y_j$，则该对被称为 **完全平局对。**
  - 如果$X_i = X_j$ 且$Y_i \neq Y_j$，则该对被称为 **仅在 X 中的平局对。**
  - 如果$X_i \neq X_j$且 $Y_i = Y_j$，则该对被称为 **仅在 Y 中的平局对。**

3.计算一致对和不一致对的数量：

- 记录一致对的数量 C和不一致对的数量 D。

4.计算 Kendall 等级秩相关系数：

- **Kendall's tau-a**（适用于没有重复排名的情况）：

$$\tau_a = \frac{C - D}{\frac{1}{2}n(n-1)}$$

- **Kendall's tau-b**（考虑重复排名的情况）：

$$\tau_b = \frac{C - D}{\sqrt{(C + D + T_1)(C + D + T_2)}}$$

其中，$T1$ 是仅在 X 中存在的平局对数量，$T2$ 是仅在 Y 中存在的平局对数量。

3.2.5.3 Kendall等级秩相关系数的Python实现

```python
import numpy as np
from scipy.stats import kendalltau

# 示例数据
X = [1, 2, 3, 4, 5]
Y = [2, 1, 5, 4, 3]

# 计算 Kendall 等级秩相关系数
tau, p_value = kendalltau(X, Y)
print(f"Kendall's Tau: {tau:.4f}")
print(f"p-value: {p_value:.4f}")
```

3.2.6 三种线性相关系数区别与联系

特征	Kendall 等级秩相关系数	Spearman 秩相关系数	Pearson 相关系数
定义	衡量两个变量之间的单调关系	衡量两个变量之间的单调关系	衡量两个变量之间的线性关系
计算方法	基于成对比较的一致性	基于秩次计算的 Pearson 相关系数	协方差与标准差的比值
数据类型	连续或有序	连续或有序	连续
分布要求	无需正态分布	无需正态分布	数据需符合正态分布
对异常值的敏感性	不敏感	不敏感	敏感
自相关	自相关为 1	自相关为 1	自相关为 1
适用场景	小样本、重复值较多的情况	非线性关系、顺序数据	线性关系
优点	稳健性强	适应性强，适用于非线性	直观且计算简单
缺点	计算复杂	可能丢失信息	对异常值敏感

注意

- **共同点**:
  - 三者都是用于衡量变量间相关性的统计量。
  - Spearman 和 Kendall 均适用于非线性关系，且不要求数据分布为正态。
- **相互关系**:
  - 在数据满足线性关系和正态分布条件下，三者之间的相关性通常较高。
  - 在非线性关系或存在异常值的情况下，Spearman 和 Kendall 可能表现更好。

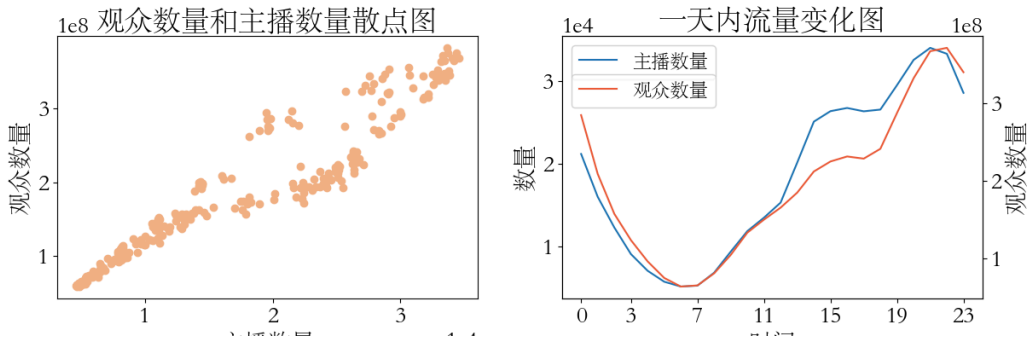
3.2.6 实践案例

案例一：

选取2019年3月6日0时至2019年3月16日24时某直播平台负载时序数据为样本，阐述二元定距变量相关分析。数据包含3个字段时间（time）、平台各时段的主播数量（streamers_count）和观众数量

```
mport pandas as pd
from matplotlib import pyplot as plt
data = pd.read_csv('D:\\PycharmProjects\\pythonProject\\chapter3\\data\\livestreaming.csv', sep=',')
df1 = data.loc[-24:, :]
df1['time'] = pd.to_datetime(data.loc[data.index[-24:], 'time'])
df1['hour'] = df1['time'].dt.hour
# 1.创建一个画布和两个子图
plt.rcParams.update({'font.size': 20, 'font.sans-serif': ['STSong'], 'axes.unicode_minus': False})
fig, axs = plt.subplots(1, 2, figsize=(14, 4))
# 2.散点图
axs[0].scatter(data['streamers_count'], data['viewers_count'], color='#F4B183')
axs[0].set(xlabel='主播数量', ylabel='观众数量', title='观众数量和主播数量散点图')
axs[0].ticklabel_format(style='sci', axis='x', scilimits=(0, 0))
# 3.折线图
df1['time'] = pd.to_datetime(df1['time'])
df1['hour'] = df1['time'].dt.hour
axs[1].plot(df1['hour'], df1['streamers_count'], label='主播数量')  # 第一条曲线
axs[1].set(xlabel='时间',ylabel='数量', title='一天内流量变化图')
axs[1].tick_params(axis='both', which='both', labelsize=20)
axs[1].ticklabel_format(style='sci', axis='y', scilimits=(0, 0))
ax2 = axs[1].twinx()  # 第二条曲线
ax2.plot(df1['hour'], df1['viewers_count'], color='#EC5D3B', label='观众数量')
ax2.set_ylabel('观众数量')
ax2.tick_params(axis='both', which='both', labelsize=20)
ax2.ticklabel_format(style='sci', axis='y', scilimits=(0, 0))
axs[1].legend(loc='upper left', fontsize=16)
ax2.legend(loc='upper left', bbox_to_anchor=(0, 0.89), fontsize=16)
plt.xticks([0, 3, 7, 11, 15, 19, 23])  # 设置x轴标签格式
plt.subplots_adjust(wspace=0.2)  # 调整子图之间的间距
plt.show()
# 4.计算皮尔逊相关系数
pearson_corr = df1['streamers_count'].corr(df1['viewers_count'])
print("Pearson相关系数:", pearson_corr)
```

输出结果如下：



其Pearson相关系数为0.952674，呈高度相关。因此网络平台在制定相关政策时，需考虑主播数量和观众数量之间的关系

案例二：

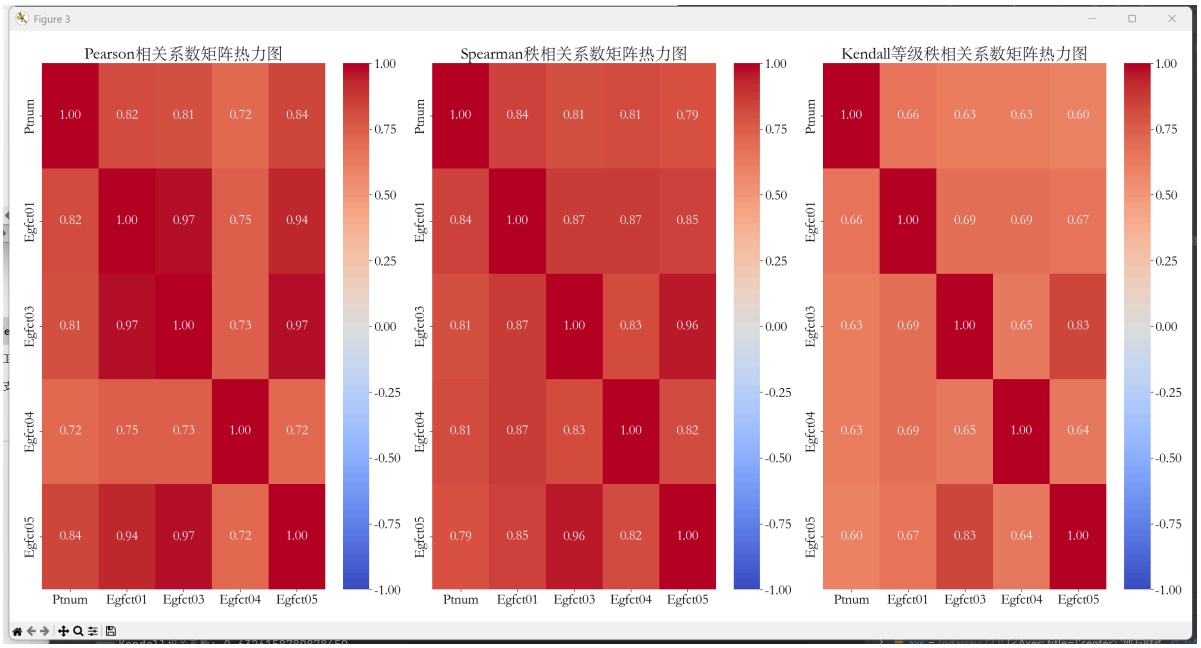
现有两份数据一份是各城市间专利转让数据，另一份是各地级市财政收支数据。以这两份原始数据为基础绘制相关系数矩阵图探讨地方财政收支（包括收入、支出、科学支出、教育支出）与专利转让之间的相关关系，并画出相关关系矩阵热力图。

```
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib
matplotlib.use('TkAgg')  # 或者 'Qt5Agg'
from scipy.stats import kendalltau, spearmanr
import seaborn as sns
# 导入数据
data1 = pd.read_excel('E:城市间专利转移情况.xlsx')  # 导入城市间转让专利数据
data3 = pd.read_excel('E:\分城市财政收支\\CRE_Gfct1.xlsx')  # 导入财政收支统计数据
city_transfer_out1= data1.groupby(['Year','Anopft'])['Ptnum'].sum().reset_index()
# 确保"Year"列为整数类型，"Ptnum"为数值型
city_transfer_out1['Year'] = pd.to_numeric(city_transfer_out1['Year'], errors='coerce')  # 将无效值转换为NaN
city_transfer_out1['Ptnum'] = pd.to_numeric(city_transfer_out1['Ptnum'], errors='coerce')
data3['Sgnyea'] = pd.to_numeric(data3['Sgnyea'], errors='coerce')
data3['Egfct04'] = pd.to_numeric(data3['Egfct04'], errors='coerce')
data3['Egfct05'] = pd.to_numeric(data3['Egfct05'], errors='coerce')
# 删除包含 NaN 值的行
city_transfer_out1.dropna(subset=['Year', 'Ptnum'], inplace=True)  # 删除 city_transfer_out1 中 Year 和 Ptnum 列中包含 NaN 的行
data3.dropna(subset=['Sgnyea', 'Egfct04','Egfct05'], inplace=True)  # 删除 data3 中 Sgnyea, Egfct04, Egfct05 列中包含 NaN 的行
# 确保转换为整数类型
city_transfer_out1['Year'] = city_transfer_out1['Year'].astype(int)
city_transfer_out1['Ptnum'] = city_transfer_out1['Ptnum'].astype(int)
data3['Sgnyea'] = data3['Sgnyea'].astype(int)
data3['Egfct04'] = data3['Egfct04'].astype(int)
data3['Egfct05'] = data3['Egfct05'].astype(int)
# 专利转让
data1_with_data3 = pd.merge(left=city_transfer_out1, right=data3, how='inner',
```

```
                                left_on=['Anopft', 'Year'], right_on=['Ctnm',
'Sgnyea'])
print(type(data1_with_data3), data1_with_data3.head(5))
# 再次剔除合并后的表格中任何包含 NaN 的行
data1_with_data3.dropna(inplace=True)
# 保存结果
file_path = 'data1_with_data3.xlsx'
data1_with_data3.to_excel(file_path, index=False)

# 热力图
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from pingouin import partial_corr
# 选择你需要的变量
selected_columns = ['Ptnum', 'Egfct01', 'Egfct03', 'Egfct04', 'Egfct05']
# 创建一个空的 DataFrame 来存储偏相关系数和Pearson相关系数
partial_corr_matrix = pd.DataFrame(index=selected_columns,
columns=selected_columns)
# 计算每对变量的相关系数
for var1 in selected_columns:
    for var2 in selected_columns:
        if var1 != var2:
            try:
                result = partial_corr(data=data1_with_data3, x=var1,
y=var2, covar=[col for col in selected_columns.columns[0:] if col != var1
and col != var2])
                partial_corr_matrix.loc[var1, var2] = result.at['pearson',
'r']
            except Exception as e:
                partial_corr_matrix.loc[var1, var2] = None
        else:
            partial_corr_matrix.loc[var1, var2] = 1.0
partial_corr_matrix = partial_corr_matrix.astype(float)
plt.subplot(1, 3, 1)  # 子图1: Pearson相关系数矩阵热力图
corr_matrix = data1_with_data3[selected_columns].corr()
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f', vmin=-1,
vmax=1)
plt.title('Pearson相关系数矩阵热力图')
plt.tight_layout()
plt.show()

# 计算每对变量的 Spearman 相关系数
spearman_corr_matrix = pd.DataFrame(index=selected_columns,
columns=selected_columns)
for var1 in selected_columns:
    for var2 in selected_columns:
        if var1 != var2:
            try:
                # 计算 Spearman 相关系数
                spearman_corr = data1_with_data3[[var1,
var2]].corr(method='spearman')
                spearman_corr_matrix.loc[var1, var2] =
spearman_corr.iloc[0, 1]  # 提取相关系数
            except Exception as e:
                spearman_corr_matrix.loc[var1, var2] = None
```

```
        else:
            spearman_corr_matrix.loc[var1, var2] = 1.0  # 自相关为1.0

# 确保矩阵为浮点型
spearman_corr_matrix = spearman_corr_matrix.astype(float)
# 绘制 Spearman 相关系数矩阵热力图
plt.subplot(1, 3, 2)  # 子图2: Pearson相关系数矩阵热力图
sns.heatmap(spearman_corr_matrix, cmap='coolwarm', annot=True, fmt='.2f', vmin=-1, vmax=1)
plt.title('Spearman秩相关系数矩阵热力图')
plt.tight_layout()
plt.show()

# 创建一个空的 DataFrame 来存储 Kendall 相关系数
kendall_corr_matrix = pd.DataFrame(index=selected_columns, columns=selected_columns)
# 计算每对变量的 Kendall 相关系数
for var1 in selected_columns:
    for var2 in selected_columns:
        if var1 != var2:
            try:
                # 计算 Kendall 相关系数
                kendall_corr = data1_with_data3[[var1, var2]].corr(method='kendall')
                kendall_corr_matrix.loc[var1, var2] = kendall_corr.iloc[0, 1]  # 提取相关系数
            except Exception as e:
                kendall_corr_matrix.loc[var1, var2] = None
        else:
            kendall_corr_matrix.loc[var1, var2] = 1.0  # 自相关为1.0

# 确保矩阵为浮点型
kendall_corr_matrix = kendall_corr_matrix.astype(float)

# 绘制 Kendall 相关系数矩阵热力图
plt.subplot(1, 3, 3)  # 子图3: Pearson相关系数矩阵热力图
sns.heatmap(kendall_corr_matrix, cmap='coolwarm', annot=True, fmt='.2f', vmin=-1, vmax=1)
plt.title('Kendall等级秩相关系数矩阵热力图')
fig.savefig('相关系数矩阵图.png', dpi=600)  # dpi:清晰度
plt.show()

```

相关系数矩阵图结果如下，通过计算Pearson、Spearman和Kendall相关系数，我们得到了三张相关系数矩阵热力图，分别展示了不同统计方法下变量间的相关性。结果显示专利转让数（Ptnum）与财政科学事业费支出（Egfct04）和教育事业费支出（Egfct05）之间的相关性较强，Pearson相关系数分别接近0.84和0.82，表明这些财政支出对促进专利转让具有显著的正向影响。专利转让数（Ptnum）与财政预算内收入（Egfct01）和支出（Egfct03）之间的相关性也呈正向，但相对较弱，Pearson相关系数分别约为0.75和0.72，说明这些财政指标与专利转让数之间存在一定的正相关关系。Spearman和Kendall相关系数进一步验证了上述发现，尽管相关系数略有不同，但总体趋势一致，表明在考虑数据的等级关系时，财政支出与专利转让数之间的关系依然稳健。



3.3 非线性相关分析

3.3.1 认识非线性相关分析

狭义上，非线性关系指的是不成比例、不能用直线表示的数量关系，通常以曲线或曲面等形式呈现。而广义上，非线性关系指的是自变量按照特殊形式变化时，产生的映射关系不符合传统线性模型的情况。非线性相关关系的特点为：当一个变量发生变化时，另一个变量的变化率不是恒定的。例如迭代关系，其上一次计算结果是下一次计算的自变量，这名下无法用通用的线性函数来描述。非线性关系与线性关系相比，更贴近客观事物的本质特征，更好地反映了复杂系统的特性，为人类提供了深入了解和建模复杂现象的工具。

3.3.2 互信息

3.3.2.1 定义

互信息（Mutual Information, MI）是一种用于衡量两个随机变量之间共享信息量的工具。它在信息论中有着非常重要的应用，用来描述通信系统的极限，同时也在概率论和统计分析中被广泛使用。互信息的数学定义为：

$$I(X;Y) = \sum_{x \in X} \sum_{y \in Y} p(x,y) \log \frac{p(x,y)}{p(x)p(y)}$$

其中：

- $p(x,y)$ 是联合概率分布。
- $p(x)$ 和 $p(y)$ 是边缘概率分布。

对于连续随机变量，上述求和变为积分。

3.3.2.2 计算方法

计算互信息的基本步骤如下：

1.数据预处理：

- 确保数据是干净的，没有缺失值，并且特征值的范围一致。

- 对于连续变量，可能需要进行离散化处理。

2.计算概率分布：

- **边缘概率分布** $p(x)$ 和$p(y)$：

$$p(x) = \frac{\text{变量 } X \text{ 中等于 } x \text{ 的观测数}}{\text{总观测数}}$$

$$p(y) = \frac{\text{变量 } Y \text{ 中等于 } y \text{ 的观测数}}{\text{总观测数}}$$

- **联合概率分布** $p(x, y)$：

$$p(x, y) = \frac{\text{变量 } X \text{ 等于 } x \text{ 且变量 } Y \text{ 等于 } y \text{ 的观测数}}{\text{总观测数}}$$

3.计算熵：

- **变量** X **的熵** $H(X)$：

$$H(X) = -\sum_{x \in X} p(x) \log p(x)$$

- **变量** Y **的熵** $H(Y)$：

$$H(Y) = -\sum_{y \in Y} p(y) \log p(y)$$

- **联合熵** $H(X, Y)$：

$$H(X, Y) = -\sum_{x \in X} \sum_{y \in Y} p(x, y) \log p(x, y)$$

4.计算互信息：

- 互信息 $I(X; Y)$ 可以通过以下公式计算：

$$I(X; Y) = H(X) + H(Y) - H(X, Y)$$

- 或者，使用联合概率分布和边缘概率分布的比值：

$$I(X; Y) = \sum_{x \in X} \sum_{y \in Y} p(x, y) \log \frac{p(x, y)}{p(x)p(y)}$$

3.3.2.3 优缺点

- **互信息的优点**

1.捕捉非线性关系

互信息不仅能够衡量线性关系，还能够捕捉非线性关系。这使得它在分析复杂数据时比传统的线性方法（如皮尔逊相关系数）更具优势。

2.无需假设数据分布

互信息不依赖于数据的分布形式，适用于各种类型的数据（连续型、离散型、混合型）。

3.通用性强

互信息可以用于衡量任何两个变量之间的关系，无论它们是连续的、离散的，还是混合的。

4.可解释性

互信息的值为非负数，值越大表示变量之间的依赖关系越强，直观且易于解释。

5.适用于高维数据

互信息可以扩展到多变量情况（如条件互信息），用于分析高维数据中的复杂依赖关系。

- **互信息的缺点**

1.计算复杂度高

对于连续变量，互信息的计算需要估计概率密度函数，计算复杂度较高，尤其是在高维数据中。

2.对数据量敏感

互信息的估计精度依赖于样本量。当样本量较小时，估计结果可能不准确。

3.对噪声敏感

数据中的噪声可能会影响互信息的计算结果，尤其是在低信噪比的情况下。

4.难以处理高维数据

虽然互信息可以扩展到多变量情况，但在高维数据中，计算和解释互信息变得非常复杂。

5.需要离散化处理

对于连续变量，通常需要将其离散化以计算互信息，而离散化的方式可能会影响结果。

3.3.2.4 应用场景

1.特征选择

在机器学习和数据挖掘中，互信息常用于特征选择。通过计算特征与目标变量之间的互信息，可以筛选出对目标变量最有影响的特征。比如在分类问题中，可以使用互信息衡量每个特征与类别标签之间的相关性，选择互信息值最高的特征。

2.图像处理

在图像处理中，互信息常用于图像配准（Image Registration），即对齐两幅图像。通过最大化两幅图像之间的互信息，可以找到最佳的对齐方式。比如，在医学图像分析中，互信息用于对齐来自不同模态的图像（如 MRI 和 CT）。

3.生物信息学

在生物信息学中，互信息被广泛用于分析基因表达数据、蛋白质相互作用网络等。比如，基因共表达网络分析：通过计算基因表达数据之间的互信息，构建基因共表达网络；蛋白质相互作用预测：通过分析蛋白质序列之间的互信息，预测蛋白质之间的相互作用。

4.自然语言处理

在自然语言处理中，互信息用于衡量词语之间的关联性，常用于词义消歧、关键词提取等任务。比如，关键词提取：通过计算词语与文档类别之间的互信息，选择最具代表性的关键词；词义消歧：通过计算词语与上下文之间的互信息，确定词语的正确含义。

5.金融分析

在金融领域，互信息可以用于分析金融变量之间的非线性关系，如股票价格与市场指数之间的关系。比如，股票关联分析：通过计算不同股票价格之间的互信息，发现它们之间的潜在关联；风险管理：通过分析金融变量之间的互信息，评估风险因素之间的依赖关系。

6.神经科学

在神经科学中，互信息被用于分析神经元活动之间的依赖关系，研究神经网络的编码和解码机制。比如，神经元活动分析：通过计算不同神经元之间的互信息，研究它们之间的功能连接；脑电图（EEG）分析：通过计算不同脑区信号之间的互信息，研究脑区之间的信息传递。

3.3.2.5 互信息的python实现

```python
from sklearn.feature_selection import mutual_info_classif, mutual_info_regression
import numpy as np

# 示例数据
X = np.random.rand(100, 5)  # 100个样本，5个特征
y_classification = np.random.randint(0, 2, size=100)  # 分类目标变量（二分类）
y_regression = np.random.rand(100)  # 回归目标变量

# 分类任务中的互信息计算
mi_classification = mutual_info_classif(X, y_classification, random_state=42)
print("分类任务中，每个特征与目标变量的互信息：", mi_classification)

# 回归任务中的互信息计算
mi_regression = mutual_info_regression(X, y_regression, random_state=42)
print("回归任务中，每个特征与目标变量的互信息：", mi_regression)
```

3.3.3 最大信息系数

3.3.3.1 定义

最大信息系数（Maximal Information Coefficient，MIC）是一种用于衡量两个变量之间关系强度的统计量，特别适用于检测非线性关系。MIC 的设计目标是能够公平地对待各种类型的关系（线性、非线性、周期性等），并提供一个介于 0 和 1 之间的分数，其中 1 表示最强的关系，0 表示没有关系。

MIC基于互信息（Mutual Information, MI）的概念。互信息是一种衡量两个变量之间依赖性的指标，能够捕捉线性和非线性关系。然而，直接使用互信息的缺点是它对数据分布的划分方式敏感。为了解决这个问题，MIC通过以下步骤计算：

1. **网格划分**：将数据的二维空间划分为不同的网格。
2. **计算互信息**：在每个网格划分下，计算两个变量的互信息。
3. **归一化**：对不同网格划分下的互信息进行归一化处理，找到最大值。

MIC的核心思想是：**通过搜索所有可能的网格划分，找到最能反映变量关系的划分方式，并计算其归一化互信息。**

3.3.3.2 计算方法

MIC的计算过程可以分为以下几个步骤：

步骤1：网格划分

- 将数据的二维空间划分为$m \times n$的网格。
- 网格的划分方式可以是等宽划分或基于数据分布的划分。

步骤2：计算互信息

- 对于每个网格划分，计算两个变量的互信息：

$$I(X,Y;G)=\sum_{x\in X}\sum_{y\in Y}p(x,y)\log\frac{p(x,y)}{p(x)p(y)}$$

其中，$p(x,y)$ 是联合概率分布，$p(x)$ 和 $p(y)$ 是边缘概率分布。

步骤3：归一化

- 对互信息进行归一化处理，使其值在0到1之间：

$$\mathrm{MIC}(X,Y)=\max_{|G|<B(n)}\frac{I(X,Y;G)}{\log\min(|G_X|,|G_Y|)}$$

其中，$|GX|$和$|GY|$是网格在 X 和 Y 方向上的划分数量，$B(n)$ 是网格划分的上限（通常取 $B(n)=n^{0.6}$ ），n是样本数量。

步骤4：搜索最优划分

- 遍历所有可能的网格划分，找到使归一化互信息最大的划分方式。

3.3.3.4. 最大信息系数的Python实现

```python
from minepy import MINE
import numpy as np

# 生成示例数据
np.random.seed(42)
x = np.random.uniform(0, 1, 1000)
y = x**2 + np.random.normal(0, 0.1, 1000)

# 计算MIC
mine = MINE()
mine.compute_score(x, y)
print("MIC:", mine.mic())
```

3.3.3.4. MIC的优缺点

优点

1. **捕捉复杂关系**：能够检测线性和非线性关系。MIC 不仅能够检测线性关系，还能够识别非线性关系、周期性关系以及其他复杂模式。这使得它在分析真实世界数据时比传统的线性方法（如皮尔逊相关系数）更具优势。
2. **公平性**：对不同类型的关系具有相似的敏感性。无论是线性关系、指数关系还是周期性关系，MIC 都能给出一个公平的度量值，而不会对某种特定类型的关系产生偏好。
3. **通用性**：适用于各种数据分布和关系类型。MIC 适用于各种数据类型和分布形式，无需对数据的分布做出假设。无论是连续变量、离散变量，还是混合变量，MIC 都可以直接应用。
4. **可解释性**：结果值在0到1之间，易于解释。

缺点

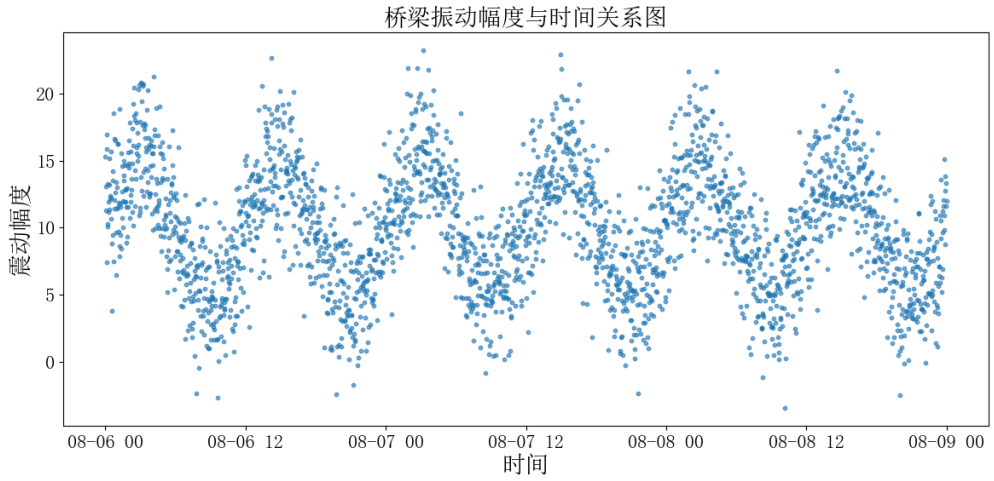
1. **计算复杂度高**：由于需要遍历所有可能的网格划分，计算时间较长。
2. **对噪声敏感**：数据中的噪声可能影响结果。如果数据中存在大量噪声，MIC 的结果可能会受到影响，导致误判变量之间的关系。
3. **参数选择**：网格划分的上限$B(n)$需要根据数据规模进行调整。

3.3.4实践案例

本案例选取2020年8月6日0时至2020年8月9日24时的某沿海桥梁潮汐震动时序作为样本，数据集包含时间（timestamp）、桥梁震动幅度（amplitude）和桥梁震动频率（frequency）。现要探究抄袭周期和桥梁震动幅度的相关性

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import pearsonr, spearmanr, kendalltau
from minepy import MINE
from sklearn.feature_selection import mutual_info_regression
# 1.加载数据集
vibration_data =
pd.read_csv('D:\\PycharmProjects\\pythonProject\\chapter3\\data\\bridge.csv'
)
# 2.确保时间戳被正确解析为datetime对象
vibration_data['timestamp'] = pd.to_datetime(vibration_data['timestamp'])
# 3.可视化振动幅度与时间的关系
plt.figure(figsize=(14, 6))
plt.rcParams['font.sans-serif'] = 'SimSun'  # 设置字体为宋体
plt.scatter(vibration_data['timestamp'], vibration_data['amplitude'],
alpha=0.6, s=10)
plt.title('桥梁振动幅度与时间关系图', fontsize=20)
plt.xlabel('时间', fontsize=20)
plt.ylabel('震动幅度', fontsize=20)
plt.grid(False)  # 去掉网格
plt.tick_params(axis='both', which='major', labelsize=16)  # 设置刻度标签字体大
小
plt.show()
# 4.计算相关系数
t_start = vibration_data['timestamp'].min()  # 获取数据集中的最早时间
time_in_seconds = (vibration_data['timestamp'] -
t_start).dt.total_seconds().values
mutual_info = mutual_info_regression(np.expand_dims(time_in_seconds,
axis=1), vibration_data['amplitude'].values)[0]  # 计算互信息
mine = MINE(alpha=0.6, c=15)
mine.compute_score(time_in_seconds, vibration_data['amplitude'].values)
mic_value = mine.mic()  # 计算MIC
pearson_corr, pearson_p_value = pearsonr(time_in_seconds,
vibration_data['amplitude'].values)  # Pearson相关
spearman_corr, spearman_p_value = spearmanr(time_in_seconds,
vibration_data['amplitude'].values)  # Spearman相关
kendall_corr, kendall_p_value = kendalltau(time_in_seconds,
vibration_data['amplitude'].values)  # Kendall相关
print("互信息:", mutual_info)
print("MIC值:", mic_value)
print("Pearson相关系数:", pearson_corr)
print("Spearman相关系数:", spearman_corr)
print("Kendall相关系数:", kendall_corr)
```

输出结果如下：




```
互信息: 0.4400620234406931
MIC值: 0.5184138187959586
```

```
Python 控制台 × 3-3 nonlinear × 3-3 nonlinear (1) ×
Pearson相关系数: -0.09501360828617339
Spearman相关系数: -0.09623069749917582
Kendall相关系数: -0.06455328328288883

In [3]:
```

从输出结果可以看出，潮汐周期和桥梁震动幅度之间的互信息和MIC值分为0.440和0.518，两者存在较为显著的非相关性，这说明时间可能是影响震动幅度的一个重要因素。而线性相关系数的结果分别为-0.095、-0.096、-0.064，说明数据之间呈现非线性相关性，尤其是周期变化时，使用非线性相关系数如互信息和MIC更能准确地捕捉到数据见的相关性。

[偏相关分析的其他实践案例参考]

3.4 偏相关分析

3.4.1 认识偏相关分析

3.4.1.1定义

偏相关分析，也称净相关分析，是在剔除其他相关因素影响的条件下计算两个变量之间的相关系数，研究两个变量的相关性。其核心思想是：**在排除 Z 的影响后，X 和 Y 之间的剩余相关性。**

偏相关系数是指偏相关分析时用于度量变量间相关性的相关系数，偏相关系数的计算公式为：

$$r_{XY \cdot Z} = \frac{r_{XY} - r_{XZ} \cdot r_{YZ}}{\sqrt{(1 - r_{XZ}^2)(1 - r_{YZ}^2)}}$$

其中：r_{XY}是 X 和 Y 之间的Person相关系数，r_{XZ} 是 X 和 Z 之间的Person相关系数，r_{YZ} 是 Y 和 Z 之间的Person相关系数。

3.4.1.3 偏相关分析的优缺点

- **优点**

1.排除混杂变量的影响：偏相关分析能够控制其他变量的影响，从而更准确地揭示两个变量之间的直接关系。比如：在研究教育水平（X）和收入（Y）之间的关系时，年龄（Z）可能是一个混杂变量。通过偏相关分析，可以排除年龄的影响，更准确地衡量教育水平和收入之间的关系。

2. 适用于多变量分析：偏相关分析可以扩展到控制多个变量的情况，适用于复杂的多变量数据分析。比如：在研究基因表达量之间的关系时，可以控制多个环境因素的影响。

3. 结果易于解释：偏相关系数的取值范围为[-1,1]，与Person相关系数类似，易于理解和解释。

- **缺点**

1. 仅适用于线性关系：偏相关分析只能衡量变量之间的线性关系，无法捕捉非线性关系。

2. 对数据分布敏感：偏相关分析通常假设数据服从多元正态分布。如果数据分布偏离正态分布，结果可能不准确。

3. 难以处理高维数据：当控制变量较多时，偏相关分析的计算复杂度会增加，且结果可能不够稳定。

4. 无法处理因果关系：偏相关分析只能衡量变量之间的相关性，无法推断因果关系

3.4.1.4偏相关分析的适用场景

1. 社会科学研究

在社会科学中，偏相关分析常用于控制混杂变量，研究两个变量之间的直接关系。比如：研究教育水平与收入之间的关系，控制年龄、性别等因素；分析工作满意度与绩效之间的关系，控制工作年限、职位等因素。

2. 生物医学研究

在生物医学领域，偏相关分析用于研究变量之间的直接关联，排除其他因素的影响。比如：研究基因表达量之间的关系，控制环境因素的影响；分析药物剂量与疗效之间的关系，控制患者年龄、体重等因素。

3. 经济学研究

在经济学中，偏相关分析用于研究经济变量之间的直接关系，排除其他经济指标的影响。比如：研究通货膨胀率与失业率之间的关系，控制 GDP 增长率等因素；分析消费者支出与收入之间的关系，控制储蓄率等因素。

4. 心理学研究

在心理学中，偏相关分析用于研究心理变量之间的直接关系，排除其他心理或生理因素的影响。比如：研究压力水平与睡眠质量之间的关系，控制年龄、性别等因素；分析幸福感与社交活动之间的关系，控制收入水平等因素。

3.4.1.5 偏相关分析的Python实现

```
1
2  import numpy as np
3  import pandas as pd
4  from scipy import stats
5
6  # 生成示例数据
7  np.random.seed(42)
8  n = 100
9  X = np.random.normal(0, 1, n)
10 Z = np.random.normal(0, 1, n)
11 Y = 2 * X + 3 * Z + np.random.normal(0, 1, n)
12
13 # 计算偏相关系数
14 def partial_corr(X, Y, Z):
```

```
    # 计算残差
    res_X = stats.linregress(Z, X).resid
    res_Y = stats.linregress(Z, Y).resid
    # 计算偏相关系数
    return stats.pearsonr(res_X, res_Y)[0]

# 计算 X 和 Y 之间的偏相关系数，控制 Z
r_XY_Z = partial_corr(X, Y, Z)
print(f"偏相关系数 (X 和 Y 控制 Z): {r_XY_Z:.4f}")
```

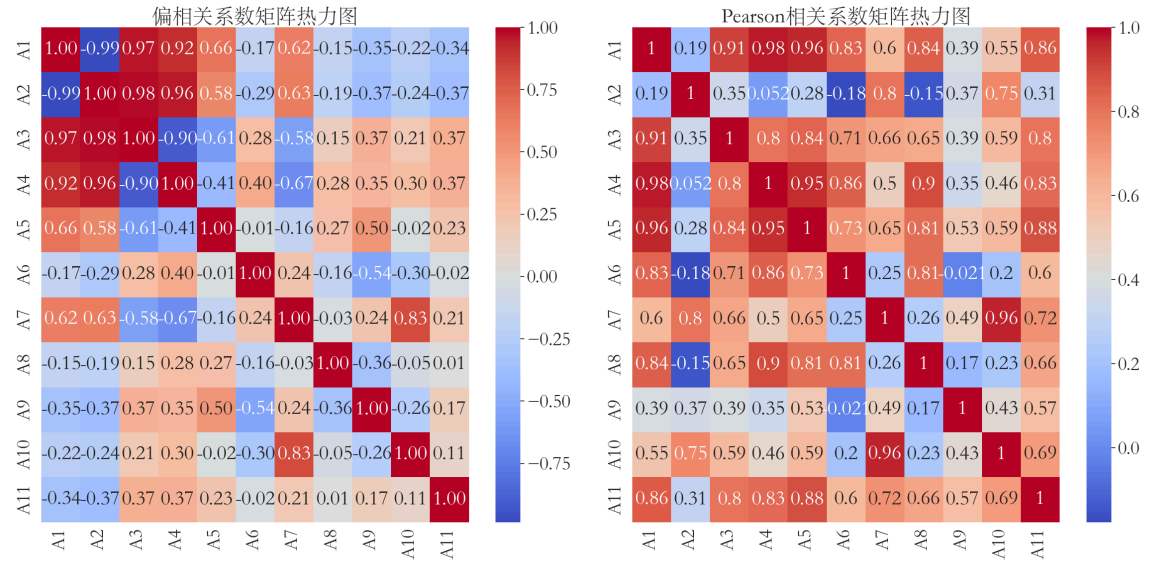
3.4.2实践案例

本案例选取2016年国内主要城市年度统计指标进行偏相关分析

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib
matplotlib.use('TkAgg')  # 或者 'Qt5Agg'
from pingouin import partial_corr
# 1. 读取数据集并处理缺失值
data = pd.read_csv('D:\\PycharmProjects\\pythonProject\\chapter3\\data\\Annual Statistical Indicators Data.csv').dropna()
data.columns = ['地区', '年份'] + ['A'+str(i) for i in range(1, 12)]  # 重命名列名
sns.set(font='STSong', font_scale=2.5)  # 设置字体和字体缩放
# 2. 计算偏相关系数并绘制热力图
plt.figure(figsize=(24, 12))
# 3.创建一个空的 DataFrame 来存储偏相关系数
partial_corr_matrix = pd.DataFrame(index=data.columns[2:], columns=data.columns[2:])
# 4.计算每对变量的偏相关系数
for var1 in data.columns[2:]:
    for var2 in data.columns[2:]:
        if var1 != var2:
            try:
                result = partial_corr(data=data, x=var1, y=var2, covar=[col for col in data.columns[2:] if col != var1 and col != var2])
                partial_corr_matrix.loc[var1, var2] = result.at['pearson', 'r']
            except Exception as e:
                partial_corr_matrix.loc[var1, var2] = None
        else:
            partial_corr_matrix.loc[var1, var2] = 1.0
partial_corr_matrix = partial_corr_matrix.astype(float)
plt.subplot(1, 2, 1)  # 子图1：偏相关系数矩阵热力图
sns.heatmap(partial_corr_matrix, cmap='coolwarm', annot=True, fmt='.2f')
plt.title('偏相关系数矩阵热力图')
plt.subplot(1, 2, 2)  # 子图2：Pearson相关系数矩阵热力图
corr_matrix = data.iloc[:, 2:].corr()
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
plt.title('Pearson相关系数矩阵热力图')
plt.tight_layout()
```

```
35 plt.show()
```

输出结果为：



3.5 距离相关分析

3.5.1 认识距离相关分析

距离相关分析是一种基于距离度量的统计方法，可以通过计算不同样本或变量之间的距离来衡量差异度或相似程度。距离相关分析根据分析对象不同，可以分为样本间距离分析和变量间距离分析。

样本间距离分析：样本间距离分析用于衡量数据集中**不同样本之间的相似性或差异性**。通过计算样本之间的距离，可以揭示样本之间的结构关系，常用于聚类分析、分类、降维等任务。

变量间距离分析：变量间距离分析用于衡量数据集中**不同变量之间的相似性或相关性**。通过计算变量之间的距离或相关性，可以揭示变量之间的依赖关系，常用于特征选择、降维、网络分析等任务。

3.5.2 距离度量方法

3.5.2.1 欧式距离

1.定义

欧氏距离是两点之间的直线距离，适用于连续型数据。

2.公式

对于两个点 $x = (x1, x2, \ldots, xn)$和$y = (y1, y2, \ldots, yn)$，欧氏距离为：

$$d = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

3.特点

- 直观且易于计算。
- 对数据的尺度敏感，需标准化处理。

4.适用场景

- 聚类分析（如 K-Means）
- 分类任务（如 KNN）
- 降维可视化（如 PCA）

5.Python实现

```python
# 方法一：手动实现

import math
def euclidean_distance(point1, point2):
    """
    计算两个点之间的欧氏距离
    :param point1: 第一个点的坐标，如 (x1, y1) 或 [x1, y1, z1]
    :param point2: 第二个点的坐标，如 (x2, y2) 或 [x2, y2, z2]
    :return: 欧氏距离
    """
    return math.sqrt(sum((a - b) ** 2 for a, b in zip(point1, point2)))

# 示例
point1 = (1, 2)
point2 = (4, 6)
distance = euclidean_distance(point1, point2)
print("欧氏距离:", distance)

# 方法二：使用 NumPy 库

import numpy as np
def euclidean_distance(point1, point2):
    """
    使用 NumPy 计算欧氏距离
    :param point1: 第一个点的坐标，如 [x1, y1]
    :param point2: 第二个点的坐标，如 [x2, y2]
    :return: 欧氏距离
    """
    return np.linalg.norm(np.array(point1) - np.array(point2))

# 示例
point1 = [1, 2]
point2 = [4, 6]
distance = euclidean_distance(point1, point2)
print("欧氏距离:", distance)

# 方法三：使用 SciPy 库

from scipy.spatial.distance import euclidean
# 示例
point1 = (1, 2)
point2 = (4, 6)
distance = euclidean(point1, point2)
print("欧氏距离:", distance)

# 方法四：使用 Python 的 math.hypot（仅限二维）

```

```python
import math
def euclidean_distance(point1, point2):
    """
    使用 math.hypot 计算二维点的欧氏距离
    :param point1: 第一个点的坐标，如 (x1, y1)
    :param point2: 第二个点的坐标，如 (x2, y2)
    :return: 欧氏距离
    """
    return math.hypot(point1[0] - point2[0], point1[1] - point2[1])

# 示例
point1 = (1, 2)
point2 = (4, 6)
distance = euclidean_distance(point1, point2)
print("欧氏距离:", distance)

```

3.5.2.2 切比雪夫距离

1.定义

切比雪夫距离是两点在各维度上差值的最大值，适用于棋盘式距离计算。

2.公式

$$D_{\text{Chebyshev}} = \max_{i=1,n}(|x_i - y_i|)$$

其中，x_i 和 y_i 分别表示两个点的第 i 维坐标。

3.特点

- 强调最大差异，忽略其他维度。
- 适用于网格状数据。

4.适用场景

- 棋盘游戏中的移动距离计算。
- 图像处理中的像素距离。

5.Python实现

```python
# 方法1：手动实现切比雪夫距离
def chebyshev_distance_manual(point1, point2):
    """
    手动实现切比雪夫距离
    :param point1: 第一个点的坐标（列表或元组）
    :param point2: 第二个点的坐标（列表或元组）
    :return: 切比雪夫距离
    """
    return max(abs(a - b) for a, b in zip(point1, point2))

# 方法2：使用 NumPy 实现切比雪夫距离
import numpy as np

```

```python
def chebyshev_distance_numpy(point1, point2):
    """
    使用 NumPy 实现切比雪夫距离
    :param point1: 第一个点的坐标（列表或 NumPy 数组）
    :param point2: 第二个点的坐标（列表或Py Num 数组）
    :return: 切比雪夫距离
    """
    return np.max(np.abs(np.array(point1) - np.array(point2)))

# 方法3：使用 SciPy 实现切比雪夫距离
from scipy.spatial.distance import chebyshev

# 示例数据
point_a = [1, 2, 3]
point_b = [4, 5, 6]

# 计算切比雪夫距离
distance_manual = chebyshev_distance_manual(point_a, point_b)
distance_numpy = chebyshev_distance_numpy(point_a, point_b)
distance_scipy = chebyshev(point_a, point_b)

# 输出结果
print(f"方法1（手动实现）：切比雪夫距离 = {distance_manual}")
print(f"方法2（NumPy 实现）：切比雪夫距离 = {distance_numpy}")
print(f"方法3（SciPy 实现）：切比雪夫距离 = {distance_scipy}")
```

3.5.2.3曼哈顿距离

1.定义

曼哈顿距离是两点在各维度上差值的绝对值之和，适用于城市街区距离计算。

2.公式

$$d(A,B) = \sum_{i=1}^{n} |x_i - y_i|$$

其中，A和B分别表示两个点。

3.特点

- 对异常值不敏感。
- 忽略了变量之间的相关性。

4.适用场景

- 高维数据或离散型数据分析
- 路径规划（如城市导航）

5.Python实现

```python
# 方法1：手动实现
def manhattan_distance_manual(point1, point2):
    """
    手动计算曼哈顿距离
    :param point1: 第一个点的坐标，如 (x1, y1)
```

```
    :param point2: 第二个点的坐标，如 (x2, y2)
    :return: 曼哈顿距离
    """
    return sum(abs(a - b) for a, b in zip(point1, point2))

# 方法2：使用 NumPy 实现
import numpy as np

def manhattan_distance_numpy(point1, point2):
    """
    使用 NumPy 计算曼哈顿距离
    :param point1: 第一个点的坐标，如 [x1, y1]
    :param point2: 第二个点的坐标，如 [x2, y2]
    :return: 曼哈顿距离
    """
    return np.sum(np.abs(np.array(point1) - np.array(point2)))

# 方法3：使用 SciPy 的 distance 函数
from scipy.spatial.distance import cityblock

def manhattan_distance_scipy(point1, point2):
    """
    使用 SciPy 的 cityblock 函数计算曼哈顿距离
    :param point1: 第一个点的坐标，如 (x1, y1)
    :param point2: 第二个点的坐标，如 (x2, y2)
    :return: 曼哈顿距离
    """
    return cityblock(point1, point2)

# 示例数据
point1 = (1, 2, 3)
point2 = (4, 6, 8)

# 计算曼哈顿距离
distance_manual = manhattan_distance_manual(point1, point2)
distance_numpy = manhattan_distance_numpy(point1, point2)
distance_scipy = manhattan_distance_scipy(point1, point2)

# 输出结果
print("手动实现的曼哈顿距离:", distance_manual)
print("使用 NumPy 的曼哈顿距离:", distance_numpy)
print("使用 SciPy 的曼哈顿距离:", distance_scipy)
```

3.5.2.4 闵可夫斯基距离

1.定义

明可夫斯基距离是欧氏距离和曼哈顿距离的推广形式，通过参数 p 控制距离的性质。

2.公式

$$D = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{\frac{1}{p}}$$

3.特点

- 当 $p = 1$ 时，退化为曼哈顿距离。
- 当$p = 2$ 时，退化为欧氏距离。
- 当 $p \to \infty$时，退化为切比雪夫距离。

4.适用场景

- 根据具体问题调整p值，适应不同的距离需求。

5.Python实现

```python
import math
import numpy as np
from scipy.spatial.distance import minkowski

# 方法1：手动实现闵可夫斯基距离
def method1_minkowski_distance(point1, point2, p):
    """
    手动实现闵可夫斯基距离
    :param point1: 第一个点的坐标
    :param point2: 第二个点的坐标
    :param p: 距离参数
    :return: 闵可夫斯基距离
    """
    return (sum(abs(a - b) ** p for a, b in zip(point1, point2))) ** (1 / p)

# 方法2：使用 NumPy 实现闵可夫斯基距离
def method2_minkowski_distance(point1, point2, p):
    """
    使用 NumPy 实现闵可夫斯基距离
    :param point1: 第一个点的坐标
    :param point2: 第二个点的坐标
    :param p: 距离参数
    :return: 闵可夫斯基距离
    """
    return np.power(np.sum(np.abs(np.array(point1) - np.array(point2)) **
p), 1 / p)

# 方法3：使用 SciPy 实现闵可夫斯基距离
def method3_minkowski_distance(point1, point2, p):
    """
    使用 SciPy 实现闵可夫斯基距离
    :param point1: 第一个点的坐标
    :param point2: 第二个点的坐标
    :param p: 距离参数
    :return: 闵可夫斯基距离
    """
    return minkowski(point1, point2, p)

# 示例
point1 = [1, 2, 3]
point2 = [4, 6, 8]
p_values = [1, 2, 3, 10]  # 不同的 p 值

```

```
print("点1:", point1)
print("点2:", point2)
print("不同 p 值下的闵可夫斯基距离：")

for p in p_values:
    print(f"\np = {p}")
    print("方法1（手动实现）:", method1_minkowski_distance(point1, point2, p))
    print("方法2（NumPy 实现）:", method2_minkowski_distance(point1, point2, p))
    print("方法3（SciPy 实现）:", method3_minkowski_distance(point1, point2, p))
```

3.5.2.5 卡方距离

1.定义

卡方距离用于衡量两个概率分布之间的差异，常用于文本分类和图像处理。

2.公式

$$D_{\chi^2}(P,Q) = \sum_{i=1}^{k} \frac{(p_i - q_i)^2}{q_i}$$

其中，P和Q分别表示两个概率分布，p_i和 q_i分别表示在第i个类别中的概率，k表示类别的数量。

3.特点

- 适用于非负数据（如直方图）。
- 对低频特征敏感。

4.适用场景

- 文本分类中的特征比较。
- 图像处理中的直方图比较。

5.Python实现

```
# 导入必要的库
import numpy as np
from scipy.spatial.distance import chi2

# 方法1：手动实现卡方距离
def method1_chi_square_distance(hist1, hist2):
    """
    手动实现卡方距离
    :param hist1: 第一个直方图或向量
    :param hist2: 第二个直方图或向量
    :return: 卡方距离
    """
    return 0.5 * sum(((a - b) ** 2) / (a + b) for a, b in zip(hist1, hist2) if a + b > 0)

# 方法1的测试
hist1 = [10, 20, 30, 40]
hist2 = [15, 25, 35, 45]
print("方法1（手动实现）:")
```

```python
print("直方图1:", hist1)
print("直方图2:", hist2)
print("卡方距离:", method1_chi_square_distance(hist1, hist2))
print("-" * 40)  # 分割线

# 方法2：使用 NumPy 实现卡方距离
def method2_chi_square_distance(hist1, hist2):
    """
    使用 NumPy 实现卡方距离
    :param hist1: 第一个直方图或向量
    :param hist2: 第二个直方图或向量
    :return: 卡方距离
    """
    hist1 = np.array(hist1)
    hist2 = np.array(hist2)
    return 0.5 * np.sum(((hist1 - hist2) ** 2) / (hist1 + hist2))

# 方法2的测试
print("方法2（NumPy 实现）:")
print("直方图1:", hist1)
print("直方图2:", hist2)
print("卡方距离:", method2_chi_square_distance(hist1, hist2))
print("-" * 40)  # 分割线

# 方法3：使用 SciPy 实现卡方距离
def method3_chi_square_distance(hist1, hist2):
    """
    使用 SciPy 实现卡方距离
    :param hist1: 第一个直方图或向量
    :param hist2: 第二个直方图或向量
    :return: 卡方距离
    """
    return chi2(hist1, hist2)

# 方法3的测试
print("方法3（SciPy 实现）:")
print("直方图1:", hist1)
print("直方图2:", hist2)
print("卡方距离:", method3_chi_square_distance(hist1, hist2))# 导入必要的库
import numpy as np
from scipy.spatial.distance import chi2

# 方法1：手动实现卡方距离
def method1_chi_square_distance(hist1, hist2):
    """
    手动实现卡方距离
    :param hist1: 第一个直方图或向量
    :param hist2: 第二个直方图或向量
    :return: 卡方距离
    """
    return 0.5 * sum(((a - b) ** 2) / (a + b) for a, b in zip(hist1, hist2)
if a + b > 0)

# 方法1的测试
hist1 = [10, 20, 30, 40]
hist2 = [15, 25, 35, 45]
```

```python
print("方法1（手动实现）:")
print("直方图1:", hist1)
print("直方图2:", hist2)
print("卡方距离:", method1_chi_square_distance(hist1, hist2))

# 方法2：使用 NumPy 实现卡方距离
def method2_chi_square_distance(hist1, hist2):
    """
    使用 NumPy 实现卡方距离
    :param hist1: 第一个直方图或向量
    :param hist2: 第二个直方图或向量
    :return: 卡方距离
    """
    hist1 = np.array(hist1)
    hist2 = np.array(hist2)
    return 0.5 * np.sum(((hist1 - hist2) ** 2) / (hist1 + hist2))

# 方法2的测试
print("方法2（NumPy 实现）:")
print("直方图1:", hist1)
print("直方图2:", hist2)
print("卡方距离:", method2_chi_square_distance(hist1, hist2))

# 方法3：使用 SciPy 实现卡方距离
def method3_chi_square_distance(hist1, hist2):
    """
    使用 SciPy 实现卡方距离
    :param hist1: 第一个直方图或向量
    :param hist2: 第二个直方图或向量
    :return: 卡方距离
    """
    return chi2(hist1, hist2)

# 方法3的测试
print("方法3（SciPy 实现）:")
print("直方图1:", hist1)
print("直方图2:", hist2)
print("卡方距离:", method3_chi_square_distance(hist1, hist2))
```

3.5.2.6 余弦相似度

1.定义

余弦相似度衡量两个向量的方向相似性，忽略大小。

2.公式

$$\text{Similarity}(A, B) = \frac{A \cdot B}{||A|| \times ||B||} = \frac{\sum_{i=1}^{n}(A_i \times B_i)}{\sqrt{\sum_{i=1}^{n} A_i^2} \times \sqrt{\sum_{i=1}^{n} B_i^2}}$$

其中，A和B是两个向量，$A \cdot B$ 表示向量的点积，$||A||$和 $||B||$分别表示向量的欧几里得范数（或大小）。

3.特点

- 取值范围为$[-1, 1]$，1 表示完全相似。
- 适用于高维稀疏数据。

4.适用场景

- 文本相似度计算。
- 推荐系统中的用户偏好比较。

5.Python实现

```python
# 方法1：手动实现余弦相似度
import math

def method1_cosine_similarity(vec1, vec2):
    """
    手动实现余弦相似度
    :param vec1: 第一个向量
    :param vec2: 第二个向量
    :return: 余弦相似度
    """
    dot_product = sum(a * b for a, b in zip(vec1, vec2))
    norm_vec1 = math.sqrt(sum(a ** 2 for a in vec1))
    norm_vec2 = math.sqrt(sum(b ** 2 for b in vec2))
    return dot_product / (norm_vec1 * norm_vec2)

# 方法1的测试
vec1 = [1, 2, 3]
vec2 = [4, 5, 6]
print("方法1（手动实现）:")
print("向量1:", vec1)
print("向量2:", vec2)
print("余弦相似度:", method1_cosine_similarity(vec1, vec2))


# 方法2：使用 NumPy 实现余弦相似度
import numpy as np

def method2_cosine_similarity(vec1, vec2):
    """
    使用 NumPy 实现余弦相似度
    :param vec1: 第一个向量
    :param vec2: 第二个向量
    :return: 余弦相似度
    """
    vec1 = np.array(vec1)
    vec2 = np.array(vec2)
    dot_product = np.dot(vec1, vec2)
    norm_vec1 = np.linalg.norm(vec1)
    norm_vec2 = np.linalg.norm(vec2)
    return dot_product / (norm_vec1 * norm_vec2)

# 方法2的测试
print("方法2（NumPy 实现）:")
```

```
print("向量1:", vec1)
print("向量2:", vec2)
print("余弦相似度:", method2_cosine_similarity(vec1, vec2))


# 方法3：使用 SciPy 实现余弦相似度
from scipy.spatial.distance import cosine

def method3_cosine_similarity(vec1, vec2):
    """
    使用 SciPy 实现余弦相似度
    :param vec1: 第一个向量
    :param vec2: 第二个向量
    :return: 余弦相似度
    """
    # SciPy 的 cosine 函数计算的是余弦距离，需要将其转换为余弦相似度
    return 1 - cosine(vec1, vec2)

# 方法3的测试
print("方法3（SciPy 实现）:")
print("向量1:", vec1)
print("向量2:", vec2)
print("余弦相似度:", method3_cosine_similarity(vec1, vec2))
```

3.5.2.7 马氏距离

1.定义

马氏距离考虑了变量之间的相关性和尺度差异，适用于多维数据。

2.公式

$$D_{\text{Mahalanobis}} = \sqrt{(x-\mu)^T \Sigma^{-1} (x-\mu)}$$

其中，x表示一个点，μ表示分布的均值，Σ表示分布的协方差矩阵。

3.特点

- 能够处理变量之间的相关性。
- 对数据分布敏感。

4.适用场景

- 异常检测。
- 多维数据的相似性度量。

5.Python实现

```
# 导入必要的库
import numpy as np
from scipy.spatial.distance import mahalanobis
import pandas as pd

# 方法1：手动实现马氏距离
def method1_mahalanobis_distance(x, y, cov):
    """
```

```
    手动实现马氏距离
    :param x: 第一个样本点
    :param y: 第二个样本点
    :param cov: 协方差矩阵
    :return: 马氏距离
    """
    diff = x - y
    inv_cov = np.linalg.inv(cov)
    distance = np.sqrt(diff.T @ inv_cov @ diff)
    return distance

# 方法1的测试
data = np.array([[1, 2], [2, 3], [3, 4], [4, 5]])
point1 = np.array([1, 2])
point2 = np.array([4, 5])
cov_matrix = np.cov(data.T)
distance1 = method1_mahalanobis_distance(point1, point2, cov_matrix)
print("方法1（手动实现）:")
print("样本点1:", point1)
print("样本点2:", point2)
print("马氏距离:", distance1)


# 方法2：使用 SciPy 实现马氏距离
def method2_mahalanobis_distance(x, y, inv_cov):
    """
    使用 SciPy 实现马氏距离
    :param x: 第一个样本点
    :param y: 第二个样本点
    :param inv_cov: 协方差矩阵的逆矩阵
    :return: 马氏距离
    """
    return mahalanobis(x, y, inv_cov)

# 方法2的测试
inv_cov_matrix = np.linalg.inv(cov_matrix)
distance2 = method2_mahalanobis_distance(point1, point2, inv_cov_matrix)
print("方法2（SciPy 实现）:")
print("样本点1:", point1)
print("样本点2:", point2)
print("马氏距离:", distance2)


# 方法3：完整流程实现（数据准备 + 马氏距离计算）
def method3_mahalanobis_distance(x, y, inv_cov):
    """
    完整流程实现马氏距离
    :param x: 第一个样本点
    :param y: 第二个样本点
    :param inv_cov: 协方差矩阵的逆矩阵
    :return: 马氏距离
    """
    diff = x - y
    return np.sqrt(diff.T @ inv_cov @ diff)

# 方法3的测试
```

```
df = pd.DataFrame(data, columns=['Feature1', 'Feature2'])
cov_matrix_df = df.cov()
inv_cov_matrix_df = np.linalg.inv(cov_matrix_df)
point1_df = df.iloc[0]
point2_df = df.iloc[1]
distance3 = method3_mahalanobis_distance(point1_df, point2_df,
inv_cov_matrix_df)
print("方法3（完整流程实现）:")
print("样本点1:", point1_df)
print("样本点2:", point2_df)
print("马氏距离:", distance3)
```

3.5.2.8 汉明距离

1.定义

汉明距离衡量两个等长字符串在相同位置上不同字符的数量，适用于离散型数据。

2.公式

$$d(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{n} \mathbb{I}(x_i \neq y_i)$$

其中，II 是指示函数， $\mathbf{x} = (x_1, x_2, \ldots, x_n)$和$\mathbf{y} = (y_1, y_2, \ldots, y_n)$是两个等长的字符串或向量。

3.特点

- 适用于二进制数据或分类数据。
- 计算简单。

4.适用场景

- 错误检测与纠正（如编码理论）。
- 基因组序列比较。

5.Python实现

```
# 方法1：手动实现汉明距离
def method1_hamming_distance(vec1, vec2):
    """
    手动实现汉明距离
    :param vec1: 第一个向量或字符串
    :param vec2: 第二个向量或字符串
    :return: 汉明距离
    """
    if len(vec1) != len(vec2):
        raise ValueError("两个输入必须具有相同的长度")
    return sum(el1 != el2 for el1, el2 in zip(vec1, vec2))

# 方法1的测试
vec1 = [1, 0, 1, 0, 1]
vec2 = [1, 1, 1, 0, 0]
print("方法1（手动实现）:")
print("向量1:", vec1)
print("向量2:", vec2)
print("汉明距离:", method1_hamming_distance(vec1, vec2))
```

```python


# 方法2：使用 NumPy 实现汉明距离
import numpy as np

def method2_hamming_distance(vec1, vec2):
    """
    使用 NumPy 实现汉明距离
    :param vec1: 第一个向量
    :param vec2: 第二个向量
    :return: 汉明距离
    """
    if len(vec1) != len(vec2):
        raise ValueError("两个输入必须具有相同的长度")
    return np.sum(np.array(vec1) != np.array(vec2))

# 方法2的测试
print("方法2（NumPy 实现）:")
print("向量1:", vec1)
print("向量2:", vec2)
print("汉明距离:", method2_hamming_distance(vec1, vec2))


# 方法3：使用 SciPy 实现汉明距离
from scipy.spatial.distance import hamming

def method3_hamming_distance(vec1, vec2):
    """
    使用 SciPy 实现汉明距离
    :param vec1: 第一个向量
    :param vec2: 第二个向量
    :return: 汉明距离
    """
    if len(vec1) != len(vec2):
        raise ValueError("两个输入必须具有相同的长度")
    # SciPy 的 hamming 函数返回的是汉明距离的比例，需要乘以向量长度
    return hamming(vec1, vec2) * len(vec1)

# 方法3的测试
print("方法3（SciPy 实现）:")
print("向量1:", vec1)
print("向量2:", vec2)
print("汉明距离:", method3_hamming_distance(vec1, vec2))
```

3.5.2.9 杰卡德相似系数

1.定义

杰卡德相似系数衡量两个集合的交集与并集的比例，适用于二元数据。

2.公式

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

其中，$|A \cap B|$ 表示集合A和集合B的交集元素个数，$|A \cup B|$ 表示集合A和集合B的并集元素个数。

3.特点

- 取值范围为 $[0, 1]$，1 表示完全相似。
- 忽略元素的具体值，只关注是否存在。

4.适用场景

- 集合相似性比较（如文档相似性）。
- 推荐系统中的用户兴趣比较。

5.Python实现

```python
# 方法1：手动实现杰卡德相似系数
def method1_jaccard_similarity(set1, set2):
    """
    手动实现杰卡德相似系数
    :param set1: 第一个集合
    :param set2: 第二个集合
    :return: 杰卡德相似系数
    """
    intersection = len(set1.intersection(set2))
    union = len(set1.union(set2))
    return intersection / union if union != 0 else 0

# 方法1的测试
set1 = {1, 2, 3, 4}
set2 = {3, 4, 5, 6}
print("方法1（手动实现）:")
print("集合1:", set1)
print("集合2:", set2)
print("杰卡德相似系数:", method1_jaccard_similarity(set1, set2))


# 方法2：使用 Python 的集合操作实现杰卡德相似系数
def method2_jaccard_similarity(set1, set2):
    """
    使用 Python 的集合操作实现杰卡德相似系数
    :param set1: 第一个集合
    :param set2: 第二个集合
    :return: 杰卡德相似系数
    """
    intersection = len(set1 & set2)
    union = len(set1 | set2)
    return intersection / union if union != 0 else 0

# 方法2的测试
print("方法2（集合操作）:")
print("集合1:", set1)
print("集合2:", set2)
print("杰卡德相似系数:", method2_jaccard_similarity(set1, set2))


# 方法3：使用 NumPy 实现杰卡德相似系数（适用于数组）
import numpy as np

```

```python
def method3_jaccard_similarity(arr1, arr2):
    """
    使用 NumPy 实现杰卡德相似系数
    :param arr1: 第一个数组
    :param arr2: 第二个数组
    :return: 杰卡德相似系数
    """
    set1 = set(arr1)
    set2 = set(arr2)
    intersection = len(set1.intersection(set2))
    union = len(set1.union(set2))
    return intersection / union if union != 0 else 0

# 方法3的测试
arr1 = np.array([1, 2, 3, 4])
arr2 = np.array([3, 4, 5, 6])
print("方法3（NumPy 实现）:")
print("数组1:", arr1)
print("数组2:", arr2)
print("杰卡德相似系数:", method3_jaccard_similarity(arr1, arr2))
```

3.5.3 总结对比

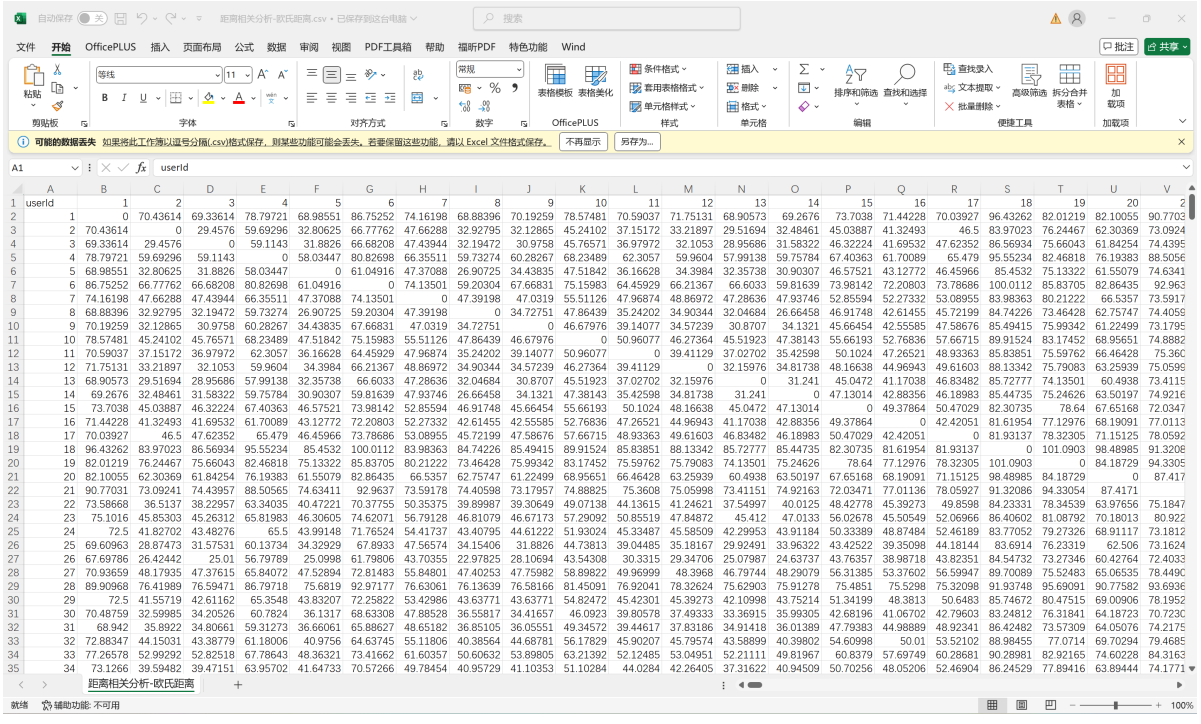
距离/相似度	特点	适用场景
欧氏距离	直线距离，对尺度敏感	聚类、分类、降维
切比雪夫距离	最大差值，适用于网格数据	棋盘游戏、图像处理
曼哈顿距离	绝对值之和，对异常值不敏感	高维数据、路径规划
明可夫斯基距离	推广形式，可调整 p 值	多种距离需求
卡方距离	衡量概率分布差异，对低频敏感	文本分类、图像处理
余弦相似度	方向相似性，适用于高维稀疏数据	文本相似度、推荐系统
马氏距离	考虑变量相关性，适用于多维数据	异常检测、多维数据分析
汉明距离	字符差异数量，适用于离散数据	错误检测、基因组比较
杰卡德相似系数	集合交集与并集比例，适用于二元数据	文档相似性、用户兴趣比较

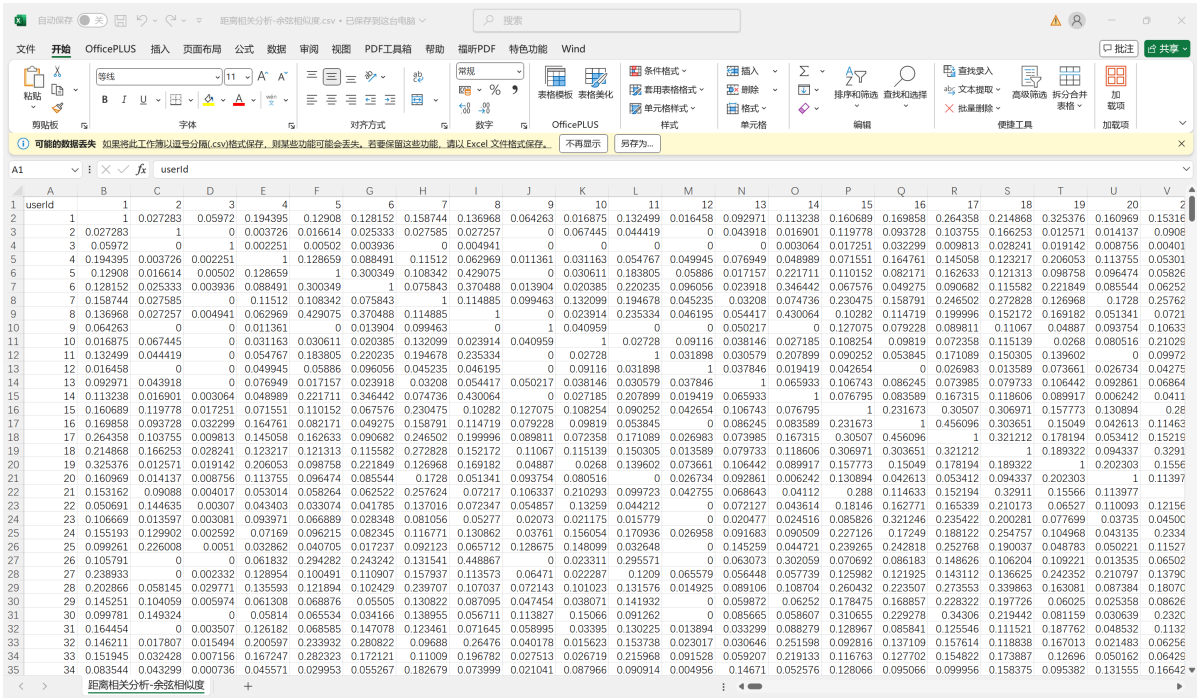
3.5.4 实践案例

```python
import pandas as pd
from scipy.spatial.distance import euclidean
from scipy.spatial.distance import cosine
# 1.导入数据
data =
pd.read_csv('D:\\PycharmProjects\\pythonProject\\chapter3\\data\\ratings.csv
')
# 2.根据userId、movieId、rating三列得到评分行
pivot_df = data.pivot(index='userId', columns='movieId', values='rating')
# 3.填充NaN值为0
```

```
pivot_df_subset = pivot_df.fillna(0)
print(pivot_df_subset)
# 4.创建新的DataFrame用于存储矩阵
matrix_df_a = pd.DataFrame(index=pivot_df_subset.index,
columns=pivot_df_subset.index)
matrix_df_b = pd.DataFrame(index=pivot_df_subset.index,
columns=pivot_df_subset.index)
# 5.填充欧式距离矩阵
for i in pivot_df_subset.index:
    for j in pivot_df_subset.index:
        # 计算用户i和用户j之间的欧式距离
        distance = euclidean(pivot_df_subset.loc[i], pivot_df_subset.loc[j])
        matrix_df_a.loc[i, j] = distance
print(matrix_df_a)
# 6.填充余弦相似度矩阵
for i in pivot_df_subset.index:
    for j in pivot_df_subset.index:
        # 计算用户i和用户j之间的余弦相似度
        similarity = 1 - cosine(pivot_df_subset.loc[i],
pivot_df_subset.loc[j])
        matrix_df_b.loc[i, j] = similarity
print(matrix_df_b)
# 7.保存结果
matrix_df_a.to_csv('D:\\PycharmProjects\\pythonProject\\chapter3\\data\\距离相关分析-欧氏距离.csv')
matrix_df_b.to_csv('D:\\PycharmProjects\\pythonProject\\chapter3\\data\\距离相关分析-余弦相似度.csv')
```

输出结果如下：





	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
1	userId	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	2
2	1	1	0.027283	0.05972	0.194395	0.12908	0.128152	0.158744	0.136968	0.064263	0.016875	0.132499	0.016458	0.092971	0.113238	0.160689	0.169858	0.264358	0.214868	0.325376	0.160969	0.15316
3	2	0.027283	1	0	0.003726	0.016614	0.025333	0.027585	0.027257	0	0.067445	0.044419	0	0.043918	0.016901	0.119778	0.093728	0.103755	0.166253	0.012571	0.014137	0.0908
4	3	0.05972	0	1	0.002251	0.00502	0.003936	0	0.004941	0	0	0	0	0	0.003064	0.017251	0.032299	0.009813	0.028241	0.019142	0.008756	0.00401
5	4	0.194395	0.003726	0.002251	1	0.128659	0.088491	0.11512	0.062969	0.011361	0.031163	0.054767	0.049945	0.076949	0.048989	0.071551	0.164761	0.145058	0.123217	0.206053	0.113755	0.05301
6	5	0.12908	0.016614	0.00502	0.128659	1	0.300349	0.108342	0.429075	0	0.030611	0.183805	0.05886	0.017157	0.221711	0.110152	0.082171	0.162633	0.121313	0.098758	0.096474	0.05826
7	6	0.128152	0.025333	0.003936	0.088491	0.300349	1	0.075843	0.370488	0.013904	0.020385	0.220235	0.096056	0.023918	0.346442	0.067576	0.049275	0.090682	0.115582	0.221849	0.085544	0.06252
8	7	0.158744	0.027585	0	0.11512	0.108342	0.075843	1	0.114885	0.099463	0.132099	0.194678	0.045235	0.03208	0.074736	0.230475	0.158791	0.246502	0.272828	0.126968	0.1728	0.25762
9	8	0.136968	0.027257	0.004941	0.062969	0.429075	0.370488	0.114885	1	0	0.023914	0.235334	0.046195	0.054417	0.430064	0.10282	0.114719	0.199996	0.152172	0.169182	0.051341	0.0721
10	9	0.064263	0	0	0.011361	0	0.013904	0.099463	0	1	0.040959	0	0	0.050217	0	0.127075	0.079228	0.089811	0.11067	0.04887	0.093754	0.10633
11	10	0.016875	0.067445	0	0.031163	0.030611	0.020385	0.132099	0.023914	0.040959	1	0.02728	0.09116	0.038146	0.027185	0.108254	0.09819	0.072358	0.115139	0.0268	0.080516	0.21029
12	11	0.132499	0.044419	0	0.054767	0.183805	0.220235	0.194678	0.235334	0	0.02728	1	0.031898	0.030579	0.207899	0.090252	0.053845	0.171089	0.150305	0.139602	0	0.09972
13	12	0.016458	0	0	0.049945	0.05886	0.096056	0.045235	0.046195	0	0.09116	0.031898	1	0.037846	0.019419	0.042654	0	0.026983	0.013589	0.073661	0.026734	0.04275
14	13	0.092971	0.043918	0	0.076949	0.017157	0.023918	0.03208	0.054417	0.050217	0.038146	0.030579	0.037846	1	0.065933	0.106743	0.086245	0.073985	0.079733	0.106442	0.092861	0.06864
15	14	0.113238	0.016901	0.003064	0.048989	0.221711	0.346442	0.074736	0.430064	0	0.027185	0.207899	0.019419	0.065933	1	0.076795	0.083589	0.167315	0.118606	0.089917	0.006242	0.0411
16	15	0.160689	0.119778	0.017251	0.071551	0.110152	0.067576	0.230475	0.10282	0.127075	0.108254	0.090252	0.042654	0.106743	0.076795	1	0.231673	0.30507	0.306971	0.157773	0.130894	0.28
17	16	0.169858	0.093728	0.032299	0.164761	0.082171	0.049275	0.158791	0.114719	0.079228	0.09819	0.053845	0	0.086245	0.083589	0.231673	1	0.456096	0.303651	0.15049	0.042613	0.11463
18	17	0.264358	0.103755	0.009813	0.145058	0.162633	0.090682	0.246502	0.199996	0.089811	0.072358	0.171089	0.026983	0.073985	0.167315	0.30507	0.456096	1	0.321212	0.178194	0.053412	0.15219
19	18	0.214868	0.166253	0.028241	0.123217	0.121313	0.115582	0.272828	0.152172	0.11067	0.115139	0.150305	0.013589	0.079733	0.118606	0.306971	0.303651	0.321212	1	0.189322	0.094337	0.3291
20	19	0.325376	0.012571	0.019142	0.206053	0.098758	0.221849	0.126968	0.169182	0.04887	0.0268	0.139602	0.073661	0.106442	0.089917	0.157773	0.15049	0.178194	0.189322	1	0.202303	0.1556
21	20	0.160969	0.014137	0.008756	0.113755	0.096474	0.085544	0.1728	0.051341	0.093754	0.080516	0	0.026734	0.092861	0.006242	0.130894	0.042613	0.053412	0.094337	0.202303	1	0.11397
22	21	0.153162	0.09088	0.004017	0.053014	0.058264	0.062522	0.257624	0.07217	0.106337	0.210293	0.099723	0.042755	0.068643	0.04112	0.288	0.114633	0.152194	0.32911	0.15566	0.113977	
23	22	0.050691	0.144635	0.00307	0.043403	0.033074	0.041785	0.137016	0.072347	0.054857	0.13259	0.044212	0	0.072127	0.043614	0.18146	0.162771	0.165339	0.210173	0.06527	0.110093	0.12156
24	23	0.106669	0.013597	0.003081	0.093971	0.066889	0.028348	0.081056	0.05277	0.02073	0.021175	0.015779	0	0.020477	0.024516	0.085826	0.321246	0.235422	0.200281	0.077699	0.03735	0.04500
25	24	0.155193	0.129902	0.002592	0.07169	0.096215	0.082345	0.116771	0.130862	0.03761	0.156054	0.170936	0.026958	0.091683	0.090509	0.227126	0.17249	0.188122	0.254757	0.104968	0.043135	0.2334
26	25	0.099261	0.226008	0.0051	0.032862	0.040705	0.017237	0.092123	0.065712	0.128675	0.148099	0.032648	0	0.145259	0.044721	0.239265	0.242818	0.252768	0.190037	0.048783	0.050221	0.11527
27	26	0.105791	0	0	0.061832	0.294282	0.243242	0.131541	0.448867	0	0.023311	0.295571	0	0.063073	0.302059	0.070692	0.086183	0.148626	0.106204	0.109221	0.013535	0.06502
28	27	0.238933	0	0.002332	0.128954	0.100491	0.110907	0.157937	0.113573	0.06471	0.022287	0.1209	0.065579	0.056448	0.057739	0.125982	0.121925	0.143112	0.136625	0.242352	0.210797	0.13790
29	28	0.202866	0.058145	0.029771	0.135593	0.121894	0.102429	0.239707	0.107037	0.072143	0.101023	0.131576	0.014925	0.089106	0.108704	0.260432	0.223507	0.273553	0.339863	0.163081	0.087384	0.18070
30	29	0.145251	0.104059	0.005974	0.061308	0.068876	0.05505	0.130822	0.087095	0.047454	0.038071	0.141932	0	0.059872	0.06252	0.178475	0.168857	0.228322	0.197726	0.06025	0.025358	0.08626
31	30	0.099781	0.149324	0	0.05814	0.065534	0.034166	0.138955	0.056711	0.113827	0.15066	0.091262	0	0.085665	0.058607	0.310655	0.229278	0.34306	0.219442	0.081159	0.030639	0.2320
32	31	0.164454	0	0.003507	0.126182	0.068585	0.147078	0.123461	0.071645	0.058995	0.03395	0.130225	0.013894	0.033299	0.088279	0.128967	0.085841	0.125546	0.111521	0.187762	0.048532	0.1132
33	32	0.146211	0.017807	0.015494	0.200597	0.233932	0.280822	0.09688	0.26476	0.040178	0.015623	0.153738	0.023017	0.030646	0.251598	0.092816	0.137109	0.157614	0.118838	0.167013	0.021483	0.06256
34	33	0.151945	0.032428	0.007156	0.167247	0.282323	0.172121	0.11009	0.196782	0.027513	0.026719	0.215968	0.091528	0.059207	0.219133	0.116763	0.127702	0.154822	0.173887	0.12696	0.050162	0.06429
35	34	0.083544	0.043299	0.000736	0.045571	0.029953	0.055267	0.182679	0.073999	0.021041	0.087966	0.090914	0.004956	0.14671	0.052576	0.128066	0.095066	0.099956	0.158375	0.095382	0.131555	0.16642